

So, you can see that for secondary index now it is quite possible that there are multiple entries for the same value of salary for example, if you look at the salary eighty thousand that is received by Professor Singh as well as Professor Kim. So, if you look at the index file of the index sequence of the salary values you will find that against 80,000 we have two entries which mark to two different records corresponding to the faculty who are drawing that salary. So, secondary index naturally has to be dense and is created when we want we feel that there is a need to quickly search on that field or that set of fields and we will discuss more about those issues later on.

(Refer Slide Time: 14:46)

The slide features a sailboat icon in the top left corner. The title 'Primary and Secondary Indices' is centered in red. To the left of the title is a vertical column of text: 'MOOCs-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018'. Below the title is a bulleted list of points:

- Indices offer substantial benefits when searching for records
- BUT: Updating indices imposes overhead on database modification --when a file is modified, every index on the file must be updated
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
 - Each record access may fetch a new block from disk
 - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access

At the bottom left is a small video thumbnail showing a man speaking. The bottom right contains navigation icons and the text 'Silberschatz, Korth and Sudarshan'.

So, indices offer substantial benefits when searching for records, but updating index impose over it; because if you create an index whenever you insert a record or a delete a record or you change the value of a field which is indexed in a record then certainly all these indices will have to be also updated and therefore, while your access time significantly reduces, because of indexing your actual update insert delete update time will increase and therefore, the indexing will have to be done carefully.

So, sequential scan using primary index is efficient, but sequential scan using secondary index is expensive. So, you will have to bring them in blocks and that will require couple of millisecond versus the amount of time that you need in the memory access. So, these are the factors that we will have to take into consideration and we will talk more about those.

(Refer Slide Time: 15:44)

Multilevel Index

- If primary index does not fit in memory, access becomes expensive
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it
 - outer index – a sparse index of primary index
 - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on
- Indices at all levels must be updated on insertion or deletion from the file

SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr, 2018
Database System Concepts - 8th Edition
26.17
©Silberschatz, Korth and Sudarshan

Now, if I have a primary index then naturally to be able to access the records I will have to first access the primary index and then do a search in that and then traverse the point at to go to the actual record in the file. So, to access the primary record we would prefer that if the primary index can actually fit into the memory. So, that I can do a in memory search like we do the binary search in a in an array.

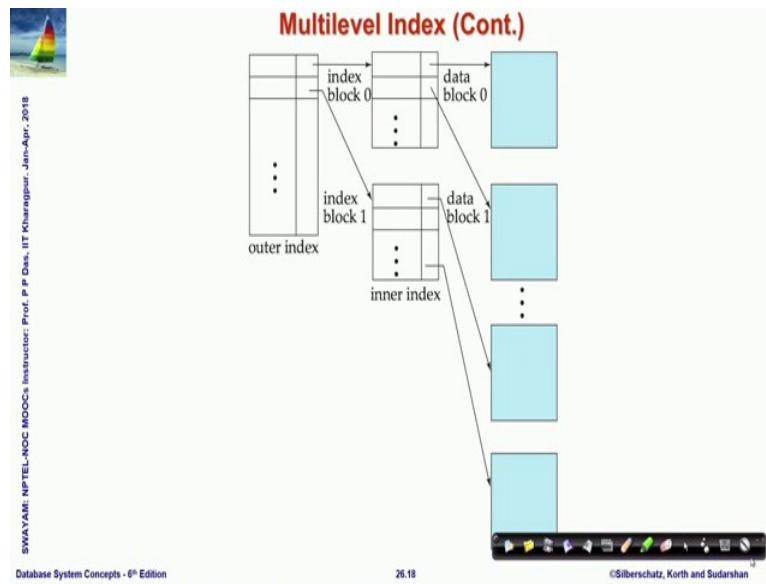
So, because if the primary index is large then that also has to exist in the disk and therefore, bringing that primary index into the memory and then searching will add to additional access cost of the disk and you have already seen in the earlier modules as. So, how these costs are expensive these accesses are expensive. So, primary index if it is not in the memory then we usually have a lot of disadvantage.

So, to take care of that if you have primary index actually is large enough. So, that it does not fit in memory then we simply apply the notion of indexing once again on the primary index file itself. So, we construct a say on the primary index we construct a past sparse index. So, which is called the outer index which is a sparse index of the primary index and in that index is the actual primary index file.

So, if now in turn the outer index the sparse index of the primary index also is too large to fit in memory then you need to do yet another level of indexing on it and. So, on till you come to a index of list which fits into a memory can be directly accessed. So, the cost for that so, this is what is called a multi level indexing.

Because, you are following multiple levels for indexing and the cost naturally of update insert delete increases; because now all of these multiple levels of indices will have to be updated.

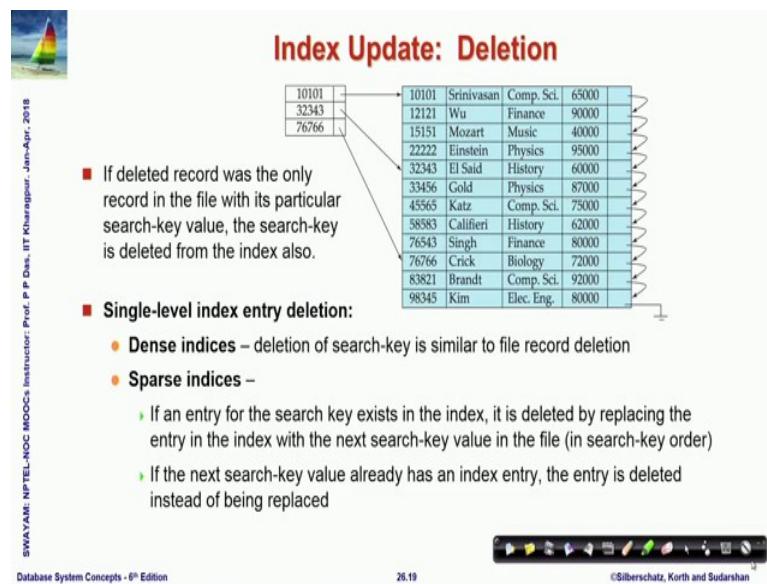
(Refer Slide Time: 17:48)



When you do an update so, this is what is a view of the multi level index you can see the outer index which is sparsely index and index those lead to different blocks of primary index of the of the inner index which is the primary index and then you traverse to the specific block where your record is expected. So, with this you since your outer index are in memory. So, you need to do one disk fetch for finding out the inner index block which should be one disk page or disk block one access and then based on that to find another access to for the block in which the record exists.

So, with this you would be able to manage with to block accesses in this case and that is how the multiple this would not have been possible if in this case you would not have done the sparse outer index, because you would have required the different parts of the inner index of the primary index to be fetched repeatedly from the disk till you act could actually find the final result in that.

(Refer Slide Time: 18:59)



The slide title is "Index Update: Deletion". It features a small sailboat icon in the top left and a decorative footer bar at the bottom. The main content includes a table of data and a detailed list of deletion rules.

Table Data:

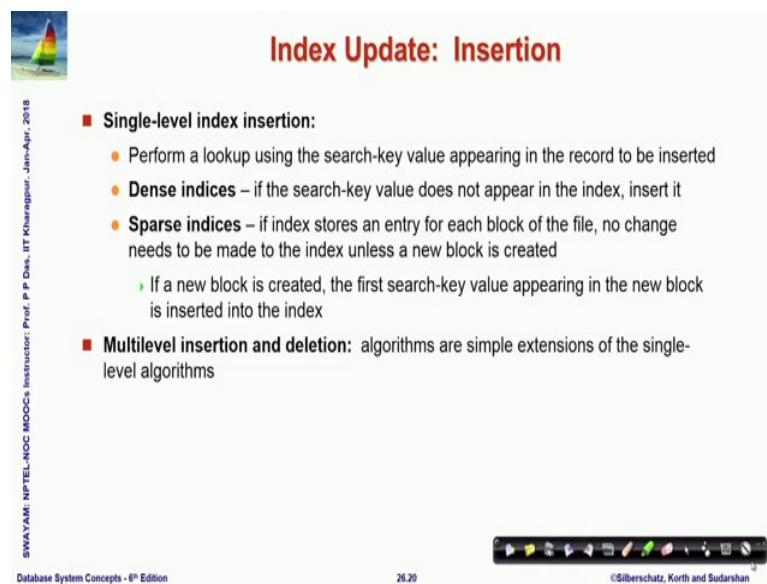
Index	Name	Subject	Score
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

List of Rules:

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.
- Single-level index entry deletion:
 - Dense indices – deletion of search-key is similar to file record deletion
 - Sparse indices –
 - ▶ If an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order)
 - ▶ If the next search-key value already has an index entry, the entry is deleted instead of being replaced

So, updating the index particularly if you do deletion then the if it is a dense index then the deletion of the search key is similar to deletion of the actual record in the file because it is dense if these parts, then naturally you will have to take care of some of the cases, because if it falls within a range then just deleting it would not matter, but if it falls on the boundary where it is actually sparsely indexed then that will have to be appropriately updated.

(Refer Slide Time: 19:32)



The slide title is "Index Update: Insertion". It features a small sailboat icon in the top left and a decorative footer bar at the bottom. The main content includes a table of data and a detailed list of insertion rules.

Table Data:

Index	Name	Subject	Score
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

List of Rules:

- Single-level index insertion:
 - Perform a lookup using the search-key value appearing in the record to be inserted
 - Dense indices – if the search-key value does not appear in the index, insert it
 - Sparse indices – if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created
 - ▶ If a new block is created, the first search-key value appearing in the new block is inserted into the index
- Multilevel insertion and deletion: algorithms are simple extensions of the single-level algorithms

Similar thing will have to be taken care of in terms of insertion. So, first you will have to look up to find out where the record needs to be inserted and then if it is a dense index if the search key does not appear in the index then you will have to insert it in case of sparse index you will have to do the additional care that whether it already belongs to the range and or whether if it will have to be a new block has to be created then it has to be also entered in terms of the sparse index and if you have multi level indexing then insertion deletion will be extensions of these basic algorithms.

(Refer Slide Time: 20:14)

The slide has a header 'Secondary Indices' in red. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list:

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition
 - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
 - Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
- We can have a secondary index with an index record for each search-key value

At the bottom left, there is a video player showing a person speaking. The video player interface includes controls for volume, brightness, and other media functions. The bottom right corner shows the text '©Silberschatz, Korth and Sudarshan'.

For secondary indices frequently we want to find all records where certain value in a certain field which is not the search key or the; of the primary index satisfy some condition. And, we can have a secondary index with an index record for each of this search key values depending on what we expect what we would think or likely fields on which more search will be done.

(Refer Slide Time: 20:45)

Module Summary

- Appreciated the reasons for indexing database tables
- Understood Indexed Sequential Access Mechanism (ISAM) and associated notions of the ordered indexes

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

26.22

©Silberschatz, Korth and Sudarshan

Or more range queries will be done. So, to summarize we have taken a brief look at the basic reasons for indexing database tables which is to speed up the process of access insert and delete and we have seen that primarily indexing primarily focuses on speeding up the access process.

So, in that maintenance of indexing whereas, insertion and deletion might have some additional overhead, but since insertion and deletion both inherently needs access to be done because you can insert only when you have tried to access and have not found exactly where the record should occur or for deletion; obviously, you need to first find the record to be able to delete it.

So, even though it is focused indexing is focused on improving the access time it actually improves the time for access insert delete all of that, but we will have to keep in mind that in the process there are certain overheads of index update for insert and delete that will have to be kept as a minimal and the additional storage requirement will also have to be kept as a least overhead. So, with this we will close this module we have taken the basic look at the index sequential access mechanism this is called index sequential access mechanism associated with different database index files.

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

Lecture - 28
Indexing and Hashing/2: Indexing/2

Welcome to module 27 of Database Management Systems. We are discussing indexing and hashing mechanisms in a database and this is the second in that series.

(Refer Slide Time: 00:28)

Module Recap

- Basic Concepts of Indexing
- Ordered Indices

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

PPD

Database System Concepts - 8th Edition 27.2 ©Silberschatz, Korth and Sudarshan

In the last module, we have discussed the basic requirement of indexing and we have learnt about ordered indexes using primary index.

(Refer Slide Time: 00:42)

The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHANDRASHEKAR NOC MOOCs", "Instructor: Prof. P. P. Deshpande", and "Date: Jan-Apr. 2018". The main content area contains two bullet points:

- To recap Balanced Binary Search Trees as options for optimal in-memory search data structures and understand the issues relating to external search data structures for persistent data
- To study 2-3-4 Tree as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

At the bottom left is a small video thumbnail showing a man speaking. The bottom right corner includes the text "Database System Concepts - 8th Edition", the page number "27.3", and the copyright notice "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is also present.

Which can be dense or parts and the multi level indexes. Now, in this module we would try to look for the basis of how indexing the index file structure can be very efficiently represent it. So, we will do a quick recap of our notions in algorithms goes.

Where we have talked about earlier balanced binary search trees I mean not in this particular course delivery, but I expect that you have gone through algorithms course where you have learnt about balanced binary search trees as options for optimal in memory search data structure and from that will try to understand the issues relating to external search data structures for persistent data and very specifically we will study 2-3-4 tree as a precursor to B/B+ tree which is an very efficient external data structure for databases and index tables. So, these are the two topics to cover.

(Refer Slide Time: 01:47)

PPD

Search Data Structures

How to search a key in a list of n data items?

- Linear Search: $O(n)$: Find 28 → 16 comparisons
 - Unordered items in an array – search sequentially
 - Unordered / Ordered items in a list – search sequentially

22	50	20	36	40	15	08	01	45	48	30	10	38	12	25	28	05	END
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

- Binary Search: $O(\log_2 n)$: Find 28 → 4 comparisons – 25, 36, 30, 28
 - Ordered items in an array – search by divide-and-conquer
 - 01 05 08 10 12 15 20 22 25 28 30 36 38 40 45 48 50 END

01	05	08	10	12	15	20	22	25	28	30	36	38	40	45	48	50	END
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

Binary Search Tree – recursively on left / right

©Silberschatz, Korth and Sudarshan

So, first let me quickly take you around with search data structure now I should warn you that here I am looking at we are looking at a little different kind of a problem here we are looking at search as it is performed in the algorithms course which mean that when you talk about data structures in algorithms course.

You typically talk of data structures which have two basic properties one they are volatile data structures transient that is they are created when the program starts and you operate on the data structure find queries and then as soon as your program ends they disappeared. So, they are not persistent data in contrast when you are dealing with database we are dealing with persistent data which stays even when no queries being performed.

And the second which is the consequence of the first is the data structure that you are study in algorithms work in memory. So, they work in a small limited space and they could be volatile whereas, the database the data structure required for databases has to reside in the disk. So, we have seen the tradeoff between the; on the storage hierarchy between memory and disk and other layers.

So, they will be brought to memory and then certain operations done and then possibly written back and so, on. So, there is similarity in terms of the strategies, but there is a very significant difference in that that because of the persistency the data structures that are used in the database application in the persistent data application has to work with a

very different kind of cost. So, when we talk about cost of an algorithm say a search algorithm we will say that a search algorithm or a sort algorithm has a certain complexity you saw, you know the merge sort has the complexity order **$n \log n$** and what it actually means.

Is a number of comparisons you can estimate to do it in sorting n numbers is approximately $n \log n$, but when we talk about external data structure or disk base data structure then your cost may often not be the operations in the CPU like comparison or addition or assignment your cost will shift to actually the number of disk accesses the page accesses that you have to do, because as you have already noted that the cost of a disk access is much larger couple of orders larger compared to the basic cost of different processor operations.

So, I will start with this in this module I will start talking about data structures which we are found to be efficient in memory and then we will migrate to seeing how they can be used in a with simple extension in the as external data structures as well. So, what we have if you have given a to search a key in a list of n data items what are the different choices I could make use of a linear search either items could be in an array ordered or unordered in an array or they could be on a list either ordered or unordered I can search that list sequentially and find the data item.

So, I have just shown an example here there is a couple of data items given in that array and trying to find 28, there are 16 comparisons that I need to do and we all know that we can do much better if we keep the this item sorted in terms of in an increasing order or say decreasing order here it is increasing order and then we can do a binary search divide and conquer and I can find the same value 28.

Now, in just four comparisons and from that we have come to the also know that this whole binary search order can be easily represented in terms of a binary tree structure called the binary search tree.

(Refer Slide Time: 05:44)

The slide has a header 'Search Data Structures' with a sailboat icon. On the left, there's vertical text: 'CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content includes a table comparing data structures based on worst-case time complexity:

Data Structure	Search	Insert	Delete	Remarks
Unordered Array	O(n)	O(1)	O(1)	The time to Insert / Delete an item is the time after the location of the item has been ascertained by Search.
Ordered Array	O(log n)	O(n)	O(n)	
Unordered List	O(n)	O(1)	O(1)	
Ordered List	O(n)	O(1)	O(1)	
Binary Search Tree	O(h)	O(1)	O(1)	

Below the table are three bullet points:

- Between an array and a list, there is a trade-off between search and insert/delete complexity
- For a BST of n nodes, $\log n \leq h \leq n$, where h is the height of the tree
- A BST is balanced if $h \sim O(\log n)$ – this is what we desire

At the bottom left is a video thumbnail of a professor, and at the bottom right are navigation icons.

So, if we compare worst case time here again here please keep in mind that we are talking about in memory data structure. So, here the time is primarily that of comparison. So, if you look at the different data structure like unordered array ordered array unordered list and ordered list and binary search tree and if you check the complexity of the three basic operations which we will need in that in a database application the search insert and delete you can find that the search. Usually, is order n unless you have an ordered array I mean between array and list the search is order n unless you have an ordered array and you can do a binary search.

The insert the time that I show for insert or for delete is usually order one, because when I say insert is order one what it means its that after I have been able to search an item which I need to insert after I have been able to find its position, what is the additional time that you need to actually insert that item? So, that insert cost is often for unorder array and any kind of list is order one because you can just manipulate a couple of pointers and insert that, but if you are using an ordered array to make your search efficient then to insert you need a order n insertion because at the right place you need to move the elements to the right to make the space for the new element, because you need to maintain the ordering that the elements have.

So, kind of we find that between the array and the list there is kind of a trade off in terms of if you want to make search better you have ordered array and that degrades the insert

delete complexity and vice versa. So, to take the benefit of both we the binary search tree is device to a you expect that the search will take the worst case order h cost which h is the height of the tree, because that is the maximum number of comparisons that we will need to reach that leaf node and a BST binary search tree if it is balanced then h would be of the order of $\log n$ and this is what we desire.

(Refer Slide Time: 07:58)

Balanced Binary Search Trees

- A BST is balanced if $h \sim O(\log n)$
- Balancing guarantees may be of various types:
 - Worst-case
 - AVL Tree
 - Randomized
 - Randomized BST, Skip List
 - Amortized
 - Splay
- These data structures have optimal complexity for all of search, insert and delete – $O(\log n)$. However:
 - Good for in memory operations
 - Work well for small volume of data
 - Has complex rotation and / or similar operations
 - Do not scale for external data structures

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

27.8

©Silberschatz, Korth and Sudarshan

So, if you look at BST is balanced h is of the order of $\log n$ I am not going into the theory of proving why balancing means h is of the order of $\log n$ or how this order of $\log n$ comes if you have doubts please refer to your algorithms book you will find plenty of that now that naturally now if a in the data structure if I am regularly inserting and deleting data.

Then it is not guaranteed that it will remain balanced it might for example, if I am inserting the data in a in a say in increasing order in a in a binary search tree then every time the insertion will happen on the rightmost node and if. So, that will mean that I will have along the rightmost node I will have a long chain and therefore, it become like a linear list and therefore, any search in that will not remain optimally it will take order in time.

So, there are different strategies that you have studied in terms of balancing just to remind you there are strategies which give you the best possible worst case time of $\log n$ which is the famous AVL tree there are randomized strategies in terms of randomized BST skip

list there are amortized strategies which say that I do not really care about what happens with a particular insertion search or deletion, but what I care is if I have done a large number of insert delete search operations on the array then on the average what it should be balanced on the average it should be of ordered $\log n$.

So, we have seen all of these different strategies and. So, in an order in an attempt to generalize them for external data structure we note that these are typically good for in memory operations these are good worked well; when you deal with a small volume of data I mean you may be that may still be large, but is small in the sense that the whole data fits into the memory and you can manipulate muscles the whole data in memory and it of course, all these many many of these algorithms.

Particularly, the AVL tree and quite a bit of the randomized BST have complex rotation operations that need to be performed the order different random generators need to be involved in the randomized BST or skip list. So, there are other complexities in this whole factor comparison cost is not the only cost that you have and in the simple thing is they do not scale, what external data structures they are not optimized in terms of minimizing or optimizing the disc accesses or they do not scale to millions and millions of entries and so, on.

(Refer Slide Time: 10:38)

The slide has a header '2-3-4 Trees' in red. On the left, there is a small sailboat icon and some text: 'MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list of properties of 2-3-4 trees:

- All leaves are at the same depth (the bottom level).
 - Height, h , of all leaf nodes are same
 - ▶ $h \sim O(\log n)$
 - ▶ Complexity of search, insert and delete: $O(h) \sim O(\log n)$
- All data is kept in sorted order
- Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively
- Generalizes easily to larger nodes
- Extends to external data structures

At the bottom, there is a video player showing a man speaking, the text 'Database System Concepts - 6th Edition', the number '27.10', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, you need to look at a different approach and this different approach in the day in terms of database application database structures is called B + tree and what I am going

to discuss is an in memory version of that which is a 2-3,-4 trees. So, that we can understand the basic principle of such search data structure which can work with external data with this data, but first understand them in as an in memory version.

So, what is a 2-3-4 tree a 2-3-4 tree in contrast to other BSTs or in contrast to the typical BSTs where you know every operation needs the height of the tree to be balanced because some leaves could happen at a much deeper level some could be at a much shallow level in a 2-3-4 tree all leaves always are at the same level the same depth the bottom level.

So, height h is the height of all the leaf nodes and that is guaranteed to be of order of $\log n$, if the tree has n number of nodes the complexity of search, delete and insert all are order h that is a consequence of a constant height h or or a fixed height h that given n that is maintained all data are kept in a sorted order. So, if you do a in order traversal of the tree you will get the data in the sorted order, but the (Refer Time: 12:13) difference is in contrast to the BST where every node is a binary node is a is a (Refer Time: 12:19) one key and has two children here every node either a leaf node or an internal node every node is one of the three types it can either be a 2-node or a 3-node or a 4-node.

A 2-node holds one, data 3-node once holds 2 data and 4-node holds 3 data and there they give the name get the name 2-node 3-node and 4-node based on the number of children that they can have the 2-node can have 2 children, 3-node 3 children and 4-node 4 children. They can generalized easily to larger nodes which can have a large number of different types of nodes and it extends very naturally to external data structure. So, that is with the basic about the 2- 3- 4 tree.

(Refer Slide Time: 13:05)

The slide is titled "2-3-4 Trees". It features a small sailboat icon at the top left and a "PPD" logo at the top right. A vertical sidebar on the left contains the text "CHALMANS NOC NOC INSTRUCTOR: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018". The main content area is divided into three sections:

- 2-node:** A circle containing "S" with two links labeled "Search keys < S" and "Search keys > S".
- 3-node:** A rectangle containing "S" and "L" with three links labeled "Search keys < S", "Search keys > S and < L", and "Search keys > L".
- 4-node:** A rectangle containing "S", "M", and "L" with four links labeled "Search keys < S", "Search keys > S and < M", "Search keys > M and < L", and "Search keys > L".

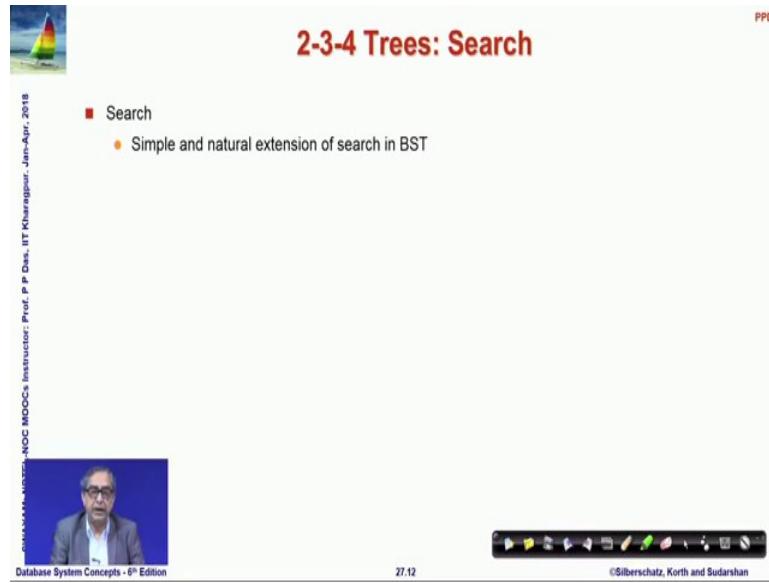
A video feed of a professor is visible in the bottom-left corner of the slide area.

So, let us just go through it in little bit more detail it uses three kinds of node as I have said and now let me just show you. So, if you are talking about a 2-node. So, this is a 2-node. So, there is one data item S and all search keys which are less than S around this side all search keys which are on greater than S around this side.

We are just for the simplicity of discussion where I am assuming that all keys in this data structure are unique. So, there is no repeated keys the repeated keys can be handled in a very easy manner. So, this is one type of node a second type of node is a 3-node which must contain two data items S we are calling it S and L and three links the first is less than S(**Search key<S**) the last is greater than L(**Search key>L**) and the middle is greater than S, but less than L(**S<Search Key<L**) which means actually what we enforce in terms of the two keys that exist at the node $S < L$.

So, values less then L go on one link values between S and L go on the middle link and values $> n$ go on the third link. Similarly, if I have a 4-node here I have three values where $S < M < L$. So, now, you can understand why the acronyms S, M and L small, medium and large. So, on and we have four links the first link gives you values **Search Key < S** second is **S<Search Key<M** third between **M< Search Key< L** and fourth **Search Key >L**. So, these are the or three different types of nodes that a 2-3-4 tree can support and if it is a leaf node then it can be it can contain either 1, 2 or 3 get items. So, let us go forward.

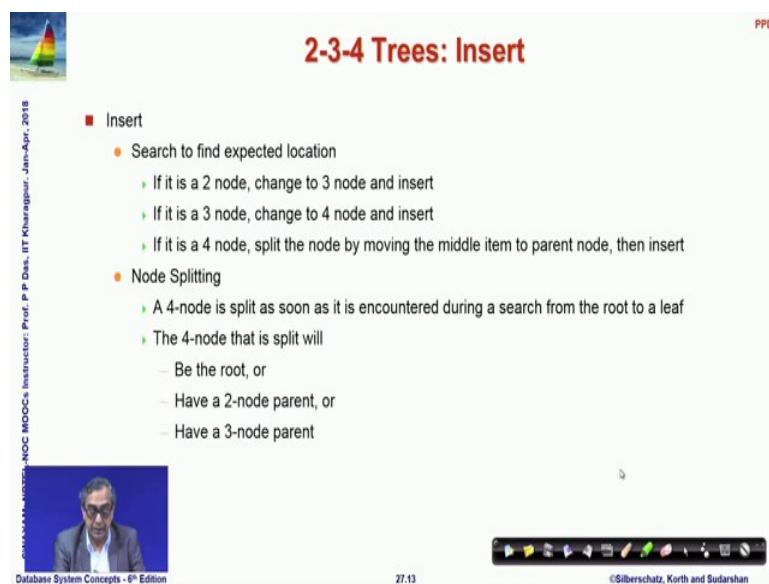
(Refer Slide Time: 15:01)



The slide title is "2-3-4 Trees: Search". It features a small sailboat icon in the top left corner and a photo of the speaker in the bottom left. The content includes a bullet point "Search" with a sub-point "Simple and natural extension of search in BST". The footer contains the text "CHANDRAKANTAPATIL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018", "Database System Concepts - 8th Edition", "27.12", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom.

Now, to search to search is a simple extension of BST you know how to search in a BST you start with the route see whether the given key to search is greater than the key at the route if it is greater you go to right if it is less you go to left and you do the same thing here for a 2-node for a 3-node all that you will need to find out is between S and L whether it **Search Key < L** then **Search Key < S** then you take the leftmost whether it is **Search Key > L** then you take the rightmost if it is **S < Search Key < L** you take the middle similar strategy you do for 4-node and search is a simple extension of the BST algorithm. So, you can suddenly work it out.

(Refer Slide Time: 15:38)



The slide title is "2-3-4 Trees: Insert". It features a small sailboat icon in the top left corner and a photo of the speaker in the bottom left. The content includes a bullet point "Insert" with a sub-point "Search to find expected location" followed by three green checkmark points: "If it is a 2 node, change to 3 node and insert", "If it is a 3 node, change to 4 node and insert", and "If it is a 4 node, split the node by moving the middle item to parent node, then insert". Another bullet point "Node Splitting" has two green checkmark points: "A 4-node is split as soon as it is encountered during a search from the root to a leaf" and "The 4-node that is split will" followed by three blue minus points: "Be the root, or", "Have a 2-node parent, or", and "Have a 3-node parent". The footer contains the text "CHANDRAKANTAPATIL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018", "Database System Concepts - 8th Edition", "27.13", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom.

Now, what we will need to do in terms of an insert, insert is very interesting. So, for insert first you search and find the expected location where you are expecting it. So, that is not there. So, you will expect. So, now, what are the possibility? Possibilities where you have found the expected location that node could be a 2-node if it is a 2-node all that you simply need to do is change that where 3-node inserts the second item if it is a 2-node it has only one item. So, just insert this new item there and making it into a 3-node good.

In the second case if you find the location if you find that it is a 3-node. So, it has two items you just change it to a 4-node and insert this is as a third item; obviously, when you insert you will have to decide has to whether where you should insert that depends on whether you are given key is greater than the key that already existed or smaller than that and so, on. Now, the question is what happens; when if you locate the place to be insert itself is already a 4-node. Now, naturally you do not have anything $>$ 4-node. So, if it is a 4-node then all that you need is to split that node you have to split the node. So, you have a 4-node.

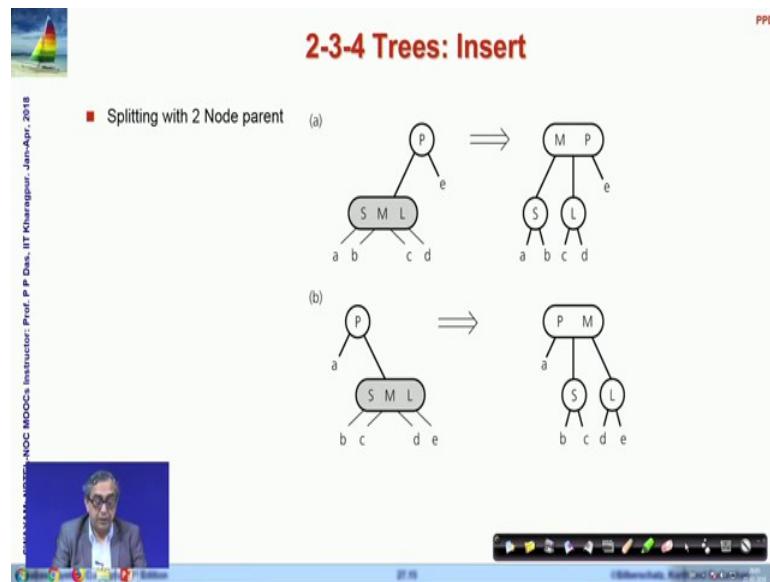
So, you have a 4-node. So, you have you have a data 1 here you have data 2 here, you have data 3 here and you are you know that the your d has to get inserted in this. So, you cannot insert it by changing the node type because this is a maximum allowed. So, all that you want to do is to change that into at different structure and that structure is called a node splitting structure. So, we will we will see in the next slide as to how this is done and we will see how different splitting sequence can happen and what will happen when a 4-node will split when it was a root or the different kinds of parents that it has let us just go there.

(Refer Slide Time: 17:42)

The slide is titled "2-3-4 Trees: Insert". It features a diagram illustrating the "Splitting at Root" operation. On the left, a 4-node (represented by a rounded rectangle) contains keys S, M, and L, with children a, b, c, and d. An arrow points to the right, where the tree has split into two 3-nodes. The left 3-node (labeled "S") contains keys a and b, with children a and b. The right 3-node (labeled "L") contains keys c and d, with children c and d. The root node "M" is shown above the two 3-nodes.

So, what we are saying is the suppose you have a 4-node which is a root now splitting that is actually doing this. So, you have to convince yourself that enough 2-3-4 tree what we have already assumed; whether I represent this or I represent this are algorithmically their equivalent. So, a single root 4-node and a such a structure of your 3- node, two nodes are equivalent why is it? So, for example, if you are looking say if you are looking for a here then it is it say it is less than s. So, you come here now if you are looking for the same a here what happens $a < S$. So, it is it **Search Key** $< M$ remember $S < M < L$ So, $a < M$. So, you take this part it is $< S$ to you come here let us take a case of c lets say c. So, if it is c you should come here, now if you check here c if it is falling on this link; that means, it is $c > M$ and $c < L$. So, $c > M$ you come here $c < L$. So, you come here. So, you reach the same link. So, you can see that actually a 2-3-4 tree is not a unique representation depending on the requirement I can replace 4-nodes in terms of other 2-nodes and actually create a new tree configuration. So, if it is at the root I can get rid of a 4-node and replace it by this equivalent tree.

(Refer Slide Time: 19:27)



Now, suppose there are the 4-node is not at the root it is somewhere else. So, what are the possibilities the; if it is not at the root it must have a parent now the parent could be a 2-node. So, if it is a 2-node then these are the two possibilities if it is a 2-node, then. So, this is a parent node which is a 2-node. So, then the 4-node could be a left child of that or it could be a right child of that if it is a left child, then we use split take the middle item and insert it in the parent and by that process a parent becomes a 3-node from a 2-node and make these become two different 2-nodes.

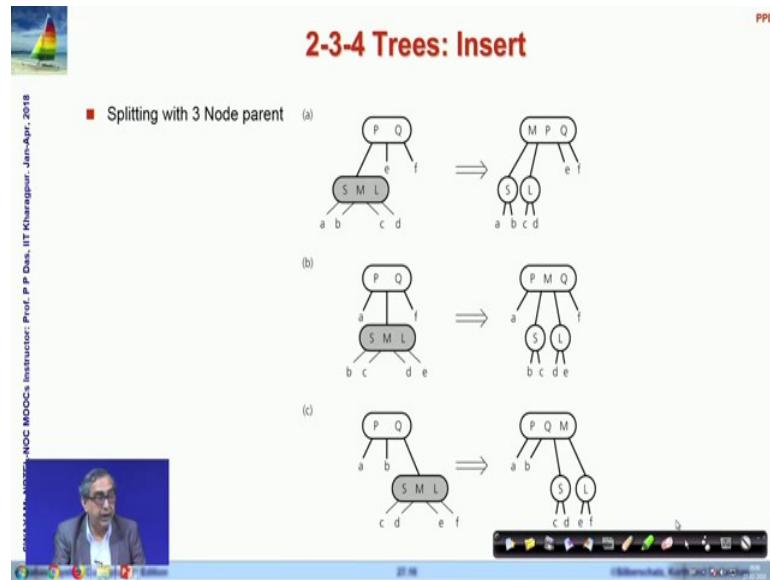
Again the way I was just explaining in the in the previous slide you can convince yourself that this structures are equivalent for example, if I am looking for d let us say if I am looking for d in this tree.

So, if I am looking for $d > L$. So, how do I arrive here now since this is a left child? So, it $d < P$, otherwise it could not have occurred on this side. So, $d < P$. So, $d < P, d > L$. So, given that if I search for d here so, $d < P$ and since it is we I also have from this the $d > M$, otherwise it would not have come to the; this third link this fourth link it would have been elsewhere. So, I know that $d > M$.

So, and $d < P$. So, if I combined this two, then d has to go on this middle ink, because it is $d > M$ and $d < P$ and then I have it is $d > L$. So, it has to be on the right. So, it comes to the right position.

So, in this way you can convince yourself in every search case that if the parent is a 2-node, then you can equivalently split the 4-node and make an insertion in the parent 2-node converted into 3-node and get rid of the 4-node altogether that lives us with.

(Refer Slide Time: 21:45)

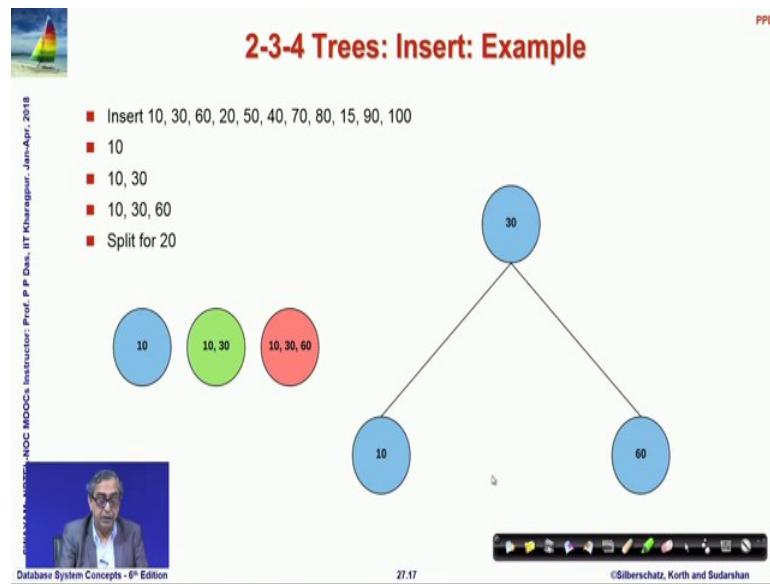


One other case which is if the parent is a 3-node naturally if it is the parent is a 3-node then there are three possibilities. Now, because your 4-node could be a left child a middle child or a right child and in every case you do the same thing you split the 4-node take the middle item put it to the parents. So, parent converts from 3-node to 4-node. Now and your rest of the split and pointed adjustments had done. So, that you get rid of this.

So, what happens in this process in and like the earlier ones here you do not get rid of the 4-nodes altogether, because in replacing one 4-node your creating another 4-node, but in the process what is happening the 4-node is moving up it is going one level up.

So, again what you will do is in recursively the new parent 4-node parent will again be split and I just split if it is its parent that is parent of the parent if that parent is also a 3-node, then that will become a 4-node this will continue till your root becomes a 4-node and we know that when root becomes a 4-node I can always change it to a configuration of 3, 2-nodes. So, in this process as we have shown that we can actually get rid of the 4-nodes in the whole of the 2-3-4 tree when it is required.

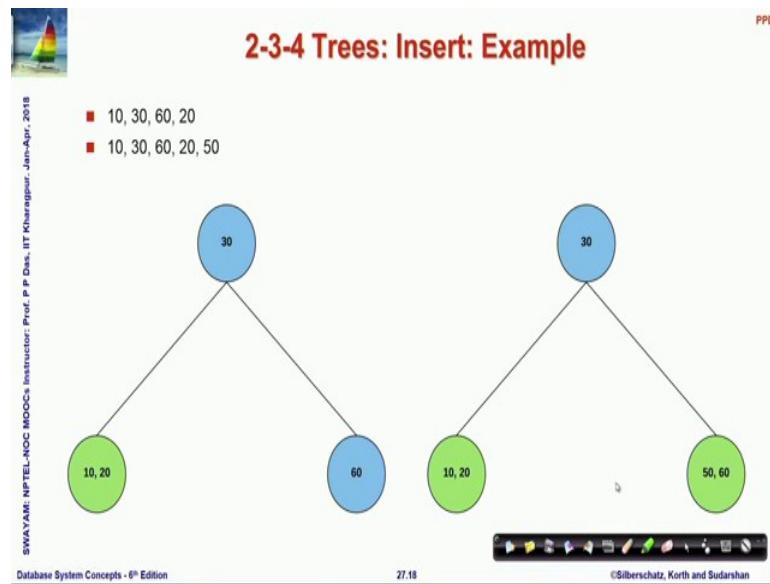
(Refer Slide Time: 23:03)



So, the basic strategy is very simple that will keep on constructing the 2-3-4 tree and whenever we come across a 4-node for the first time we will split that and rearrange. So, that I can get rid of it so, here what I show you is a basically an insert sequence over the next couple of slides where which is trying to insert the data in this following order starting with an empty 2-3-4 tree. So, we first insert 10. So, you get this then we insert 30 that is here. So, 2-node becomes. So, the here the convention that I am I am following is blue is a 2-node green is a 3-node and red is a 4-node. So, then you insert 60 it becomes a 4-node and as soon as it becomes a 4-node the next element to be entered is 20.

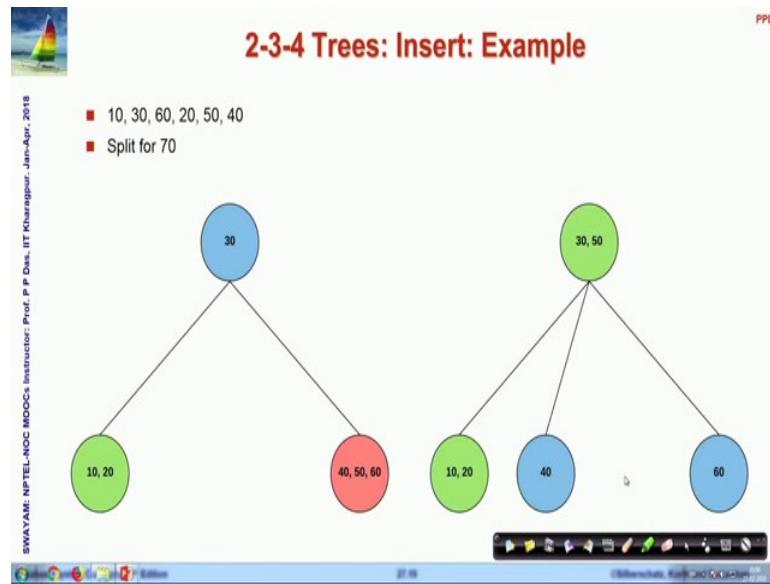
But, before that this 4-node will have to be split and this is the case of splitting at the root, because this is this has no child yet. So, you split and you get the middle moves to the top here as 30, then you have two children 10 and 60 and after the splitting you move on to actually inserting the intended 20 into it.

(Refer Slide Time: 24:19)



So, 20 gets inserted on this side 20 is inserted on this side, because a smaller than this and comes here. So, this 2-node becomes a 3-node then you insert 50 which goes on this side which goes on this side and gets inserted here.

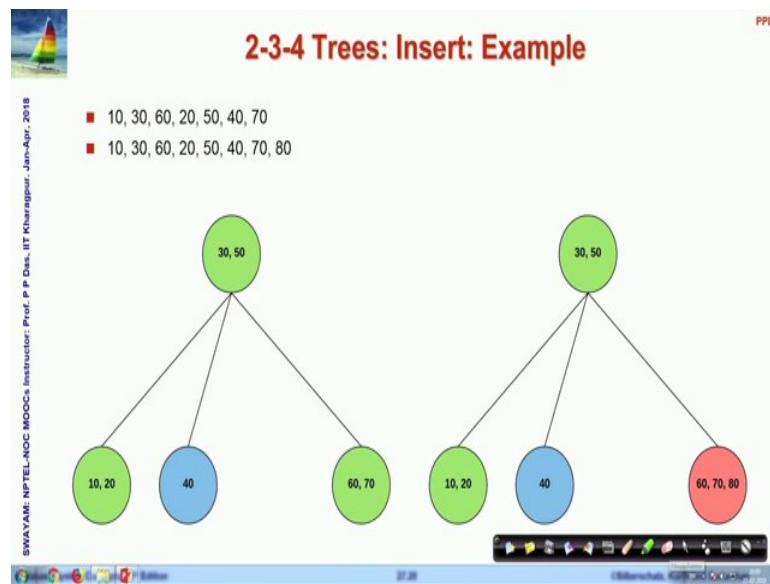
(Refer Slide Time: 24:41)



So, your insertion continues your next to insert is 40 which gets the inserted here it becomes a 4-node and as soon as it becomes a 4-node and before the next insertion of 70 can happen you need to do a split. So, you do a split.

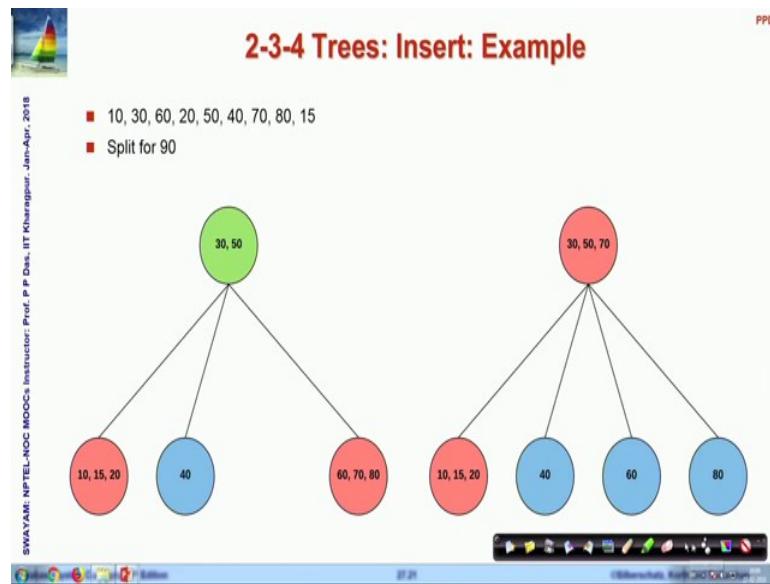
If you do a split then your 50 moves to the parent this becomes a 3-node. Now and your 40 and 60 becomes two 2-node children as this. So, it is a same information is represented, but now in this new 2-3-4 tree you do not have any 4-node. So, in you can go ahead and insert 70.

(Refer Slide Time: 25:19)



So, insert 70, 70 gets inserted here the 2-node becomes 3-node the, this is done. So, then you insert 80, 80 gets inserted here is greater than it comes here and this becomes a 4-node and. So, naturally the next would be two insert 15. So, 15 goes in on to the left. So, 15 has come in here which becomes a 4-node.

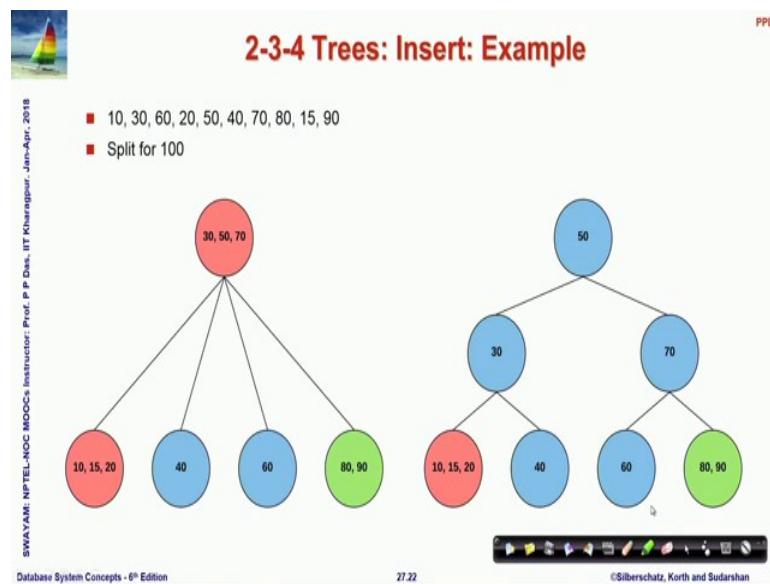
(Refer Slide Time: 25:43)



And the next is to insert 90. So, which has to get inserted here it should have got inserted here. So, this is already a 4-node. So, you need to split.

So, in split and as you split you get 40, 40 was already there you get 60 and 80 and the 70, that existed in the middle goes to the root and your root becomes a 4-node.

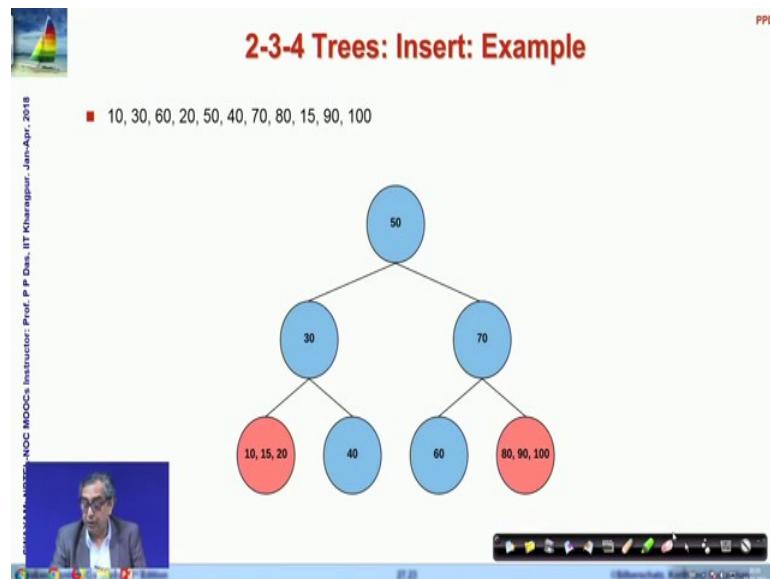
(Refer Slide Time: 26:27)



So, this is the configuration you move on to your 90 gets inserted. Now, you have to insert 100. So, you before that this your root has become a 4-node. So, you do a split. So,

as you do the split this is a new configuration that has happened and this is what has resulted from the split of this 4-node at the root.

(Refer Slide Time: 26:59)



And then naturally you can go ahead and insert 100 and 100 is now inserted. Here, you could do adjustments of changing this this 4-node into by splitting it and moving it onwards I have not shown that, but if you now go back if you just now go back on on these this whole process and you will see that specifically that we claim that all leaf nodes would be in the same level. So, you can see that initial three trees all are at the same level, then leaf and their level 0 and then when you have had this split then the leaf nodes 10 and 60 are both at level once.

So, we can see that when actually you split at the root you add one level to all the leaf nodes and that is the only time your height changes. So, beyond that you look at this the level has not changed I am going to the next the level has not changed instruments to be height 1 level does not change height 1 does not change does not change till the left here and then we have a case again of splitting a 4-node at the root.

When the one level has been added to all the; so, all the leaf nodes where at level 1 now all of them are at level 2. So, in a 2-3-4 tree you achieve this invariance of all leaf nodes being at the same level by maintaining that the only time the height changes is when you split a 4-node at the root and add one level uniformly to all of them.

And then rest of the logic is quite simple that these are here you can do actually follow the same logic as of the binary search tree analysis that if there are n items here; then the maximum height could actually be $\log n$; because we though all nodes are not binary, but the nodes that are not binary actually have more data.

So, they will be they will the height will always be $\log n$ or less than that of course, there is an issue of concluding the complexity, because now you will argue that there are 3-nodes or 4-nodes where more than one comparison is required to decide about the node, but the counter argument to that is even if that be the case even if you have along the path of the tree from the root to any leaf node.

Even if all of the nodes are of are 4-node that cannot happen as you have seen because you will keep on splitting and distributing that, but if all of them are or 4-node also then what will add is simply a factor of three two rather additional with the $\log n$ and the overall complexity remains to be $\log n$ will go.

(Refer Slide Time: 29:59)

The slide has a title '2-3-4 Trees: Delete' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading 'Delete':

- Delete
 - Locate the node n that contains the item $theItem$
 - Find $theItem$'s inorder successor and swap it with $theItem$ (deletion will always be at a leaf)
 - If that leaf is a 3-node or a 4-node, remove $theItem$
 - To ensure that $theItem$ does not occur in a 2-node
 - ▶ Transform each 2-node encountered into a 3-node or a 4-node
 - ▶ Reverse different cases illustrated for splitting

At the bottom left, there is a video player showing a man speaking, with the text 'Database System Concepts - 8th Edition'. At the bottom right, there is a navigation bar with icons and the text '27.24' and '©Silberschatz, Korth and Sudarshan'.

Now, if you have to delete you have to do very similar operations I have not shown the details, but you look at the node and then actually find the in order successor for that and swap it with the item and then you can do the other arrangements of collapsing the nodes as we have done the splitting of nodes. Now I have to do the collapsing of nodes following the same river structure and leave that as an exercise to you just work it out at home.

(Refer Slide Time: 30:28)

The slide is titled "2-3-4 Trees" in red at the top right. At the top left is a small sailboat icon. On the right side, there is some small text that appears to be "PPD".
Advantages

- All leaves are at the same depth (the bottom level): Height, $h \sim O(\log n)$
- Complexity of search, insert and delete: $O(h) \sim O(\log n)$
- All data is kept in sorted order
- Generalizes easily to larger nodes
- Extends to external data structures

Disadvantages

- Uses variety of node types – need to destruct and construct multiple nodes for converting a 2 Node to 3 Node, a 3 Node to 4 Node, for splitting etc.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
27.25
©Silberschatz, Korth and Sudarshan

So, if you look at if the 2-3-4 tree and there is a time to justify why we are doing this all leaves are the same depth which is a great advantage the height is order $\log n$ always complexity of search insert delete all are order $\log n$ all data kept in sorted order it generalizes easily to larger nodes are and extends to external data structure. So, what you mean by larger nodes, let us let us look at that a little bit before that of course, 2-3-4 tree has a major disadvantage compared to binary research trees of the other kinds because it uses a variety of node types. So, you I mean when you change from a say 2-node to a 3-node you actually if you think in programming terms you have lot of additional cost, because you need to distract the 2-node and create a 3-node.

If you split a 4-node and create 3, 2-nodes as required, then you have to distract the 4-node and create 3, 2-nodes. So, there are lot of over rights in terms of that.

(Refer Slide Time: 31:32)

The slide is titled "2-3-4 Trees" in red at the top right. At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018". The main content is a bulleted list of properties:

- Consider only one node type with space for 3 items and 4 links
 - Internal node (non-root) has 2 to 4 children (links)
 - Leaf node has 1 to 3 items
 - Wastes some space, but has several advantages for external data structure
- Generalizes easily to larger nodes
 - All paths from root to leaf are of the same length
 - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
 - A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
 - Special cases:
 - » If the root is not a leaf, it has at least 2 children.
 - » If the root is a leaf, it can have between 0 and $(n-1)$ values.
- Extends to external data structures
 - B-Tree
 - 2-3-4 Tree is a B-Tree where $n = 4$

At the bottom right, there is a set of small navigation icons.

So, what leaf if we if we just simplify that process and if you assume that, there is only one node type which has enough space for three items and four links we do not have two I mean we functionally there will be 2-node 3-node 4-node, but physically let them with a same type of node. So, any internal node can have what is specification? So, there the same type any internal node has 2 to 4 children and a leaf node has 1 to 3 items that is what all that we are saying.

So, by doing this we will waste some space, but we have several advantages particularly when you look at this for external data structure. So, what will happen is if I can think about 2-3-4 tree in that manner then I can; obviously, generalize that it is not necessary that I will have to restrict myself at 4 links, 4 children, I can do more than that.

So, in general I can say that there are a node has n children and it is each node is not a leaf not a root or a leaf will have between $n / 2$ and n children. So, put n as 4 you will find that it becomes 2-3-4 tree and a leaf node will have $(n-1) / 2$ which is 1 and for enough n being 4 and $n-1$ or 3 values which is what did you have.

So, if you just generalize from 2-3-4 to $n/2$ to n and have a container have a node container which can have either $n/2$ or $(n / 2) + 1$ or $(n / 2) + 2$ or maximum up to $n - 1$ data items and corresponding number of children then we will be very easily be able to arrange for a similar data structure which will have all the nodes all the leaf nodes at the same level and. So, this is the structure which is which extends very easily and this is a

fundamental structure of what he says a B-tree or a B + tree will see the differences shortly.

But this is a basic notion and the strategies of node splitting and node merging in case of deletion and the algorithm of insertion deletion that we have discussed here will simply get generalize in case of B-tree.

(Refer Slide Time: 33:57)

Module Summary

- Recapitulated the notions of Balanced Binary Search Trees as options for optimal in-memory search data structures
- Understood the issues relating to external data structures for persistent data
- Explored 2-3-4 Tree in depth as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jun-Apr. 2018

Database System Concepts - 8th Edition 27.27 ©Silberschatz, Korth and Sudarshan

When we go to the next module so, we have recapitulate in on the balanced binary search tree and we introduced a the notion of a 2- 3- 4 tree which is a precursor to B + tree B-tree which is which are efficient external data structures and will be covered in the next module.

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

Lecture - 28
Indexing and Hashing/3 : Indexing/3

Welcome to module 28 of Database Management Systems. We have been discussing about indexing and hashing.

(Refer Slide Time: 00:28)

PPD

Module Recap

- Balanced Binary Search Trees
- 2-3-4 Tree

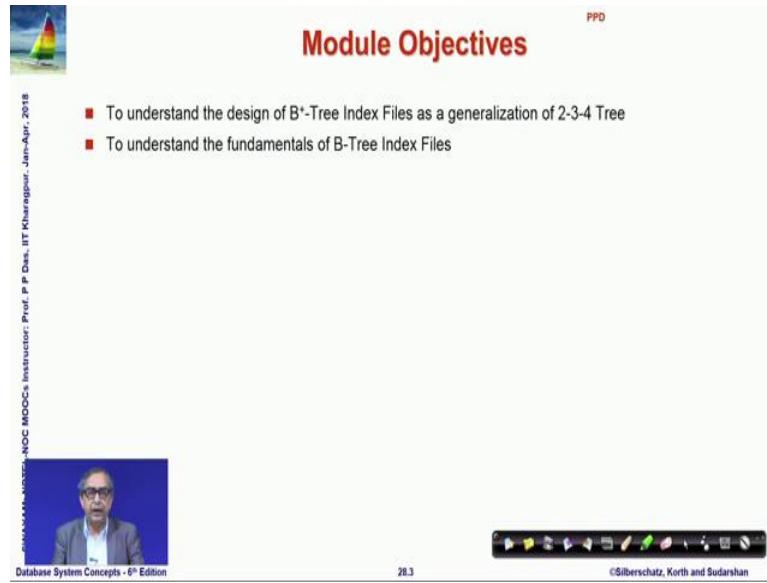
Database System Concepts - 8th Edition

28.2

©Silberschatz, Korth and Sudarshan

This is the third module; in that continuation.

(Refer Slide Time: 00:38)



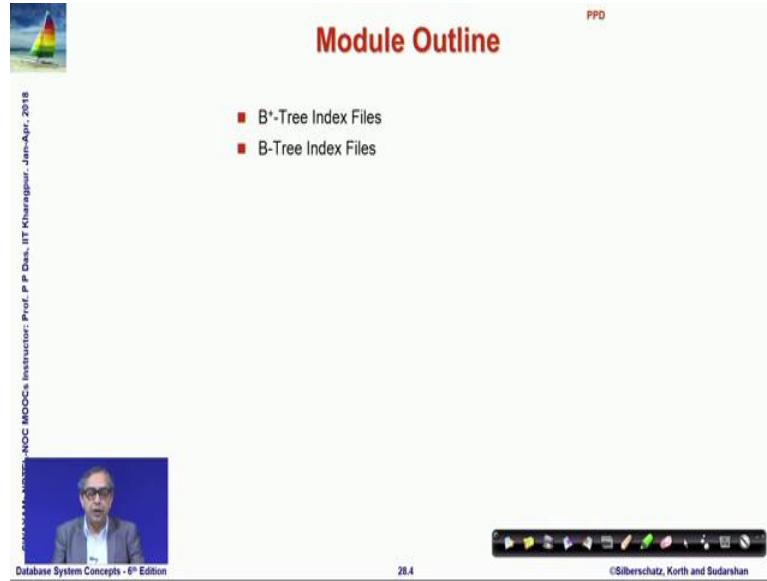
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "CHANDRAKANTAPURE-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". In the center, there is a list of objectives:

- To understand the design of B+-Tree Index Files as a generalization of 2-3-4 Tree
- To understand the fundamentals of B-Tree Index Files

At the bottom left is a video frame showing a man speaking. The bottom right contains the text "Database System Concepts - 8th Edition", the page number "28.3", and the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

In the last module we have taken a quick look at the balanced BST and specifically a and different kind of inline data structure called 2-3-4 tree, which can be of very good use in terms of understanding B+ tree, which we want to study in this module and we will also take a quick look at the B tree.

(Refer Slide Time: 00:50)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "CHANDRAKANTAPURE-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". In the center, there is a list of topics:

- B+-Tree Index Files
- B-Tree Index Files

At the bottom left is a video frame showing a man speaking. The bottom right contains the text "Database System Concepts - 8th Edition", the page number "28.4", and the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

So, now, B + tree is the main data structure is or one of the main data structures to be used for index files.

(Refer Slide Time: 01:00)

The slide has a header 'B+-Tree Index Files' with a sailboat icon. It lists the following points:

- B+-tree indices are an alternative to indexed-sequential files
- Disadvantage of indexed-sequential files
 - performance degrades as file grows, since many overflow blocks get created
 - Periodic reorganization of entire file is required
- Advantage of B+-tree index files:
 - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions
 - Reorganization of entire file is not required to maintain performance
- (Minor) disadvantage of B+-trees:
 - extra insertion and deletion overhead, space overhead
- Advantages of B+-trees outweigh disadvantages
 - B+-trees are used extensively

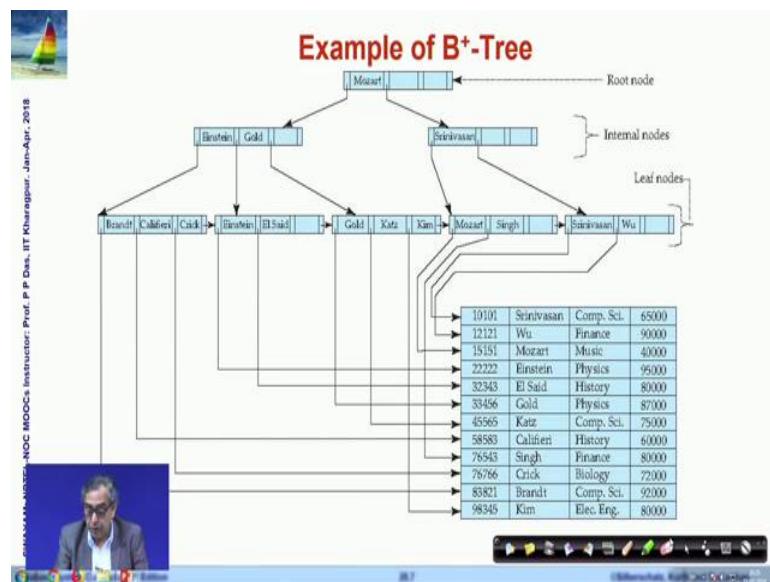
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur, Jan-Apr., 2018
Database System Concepts - 8th Edition
28.6
©Silberschatz, Korth and Sudarshan

So, B + tree has now; what we have seen we have seen the ordered indexes. We have seen the index sequential files, where you could keep the index file in a sorted manner in the primary index you could build secondary index on that and so, on, but that is not an efficient way of doing things, because the performance keeps on degrading as the file grows.

Since many overflow blocks will get created, because certainly if you are growing, then naturally you have created say sparse index on uncertain values and if there are more records in that bucket. Then naturally you need to have linked buckets. So, periodic reorganization of the entire file becomes required which is a very costly affair.

In contrast advantage of B + tree is it automatically reorganizes itself in small bits and pieces with local changes and so, on; whenever insertions and deletions happen and the reorganization of the entire file is not required for the purpose of maintenance. Of course, there are a little bit of disadvantage the extra insertion and deletion overhead exist for the small you know micro reorganization there is little bit of space over it, but in the face of the advantage that we get it outweighs the advantage is outweighs the disadvantages and B + trees are used quite extensively.

(Refer Slide Time: 02:28)



So, just recall the notion of 2-3-4 tree that we had discussed and look at this diagram. So, 2-3-4 tree have different types of node : 2 node 3 node and 4 node. So, we said that there could be a node which can be only partially filled and it has a different number of children pointing to the; conditions of how different keys are ordered in that particular node.

So, here we I show an instance of a B + tree, which is basically trying to represent this file in terms of the creating indexes. So, if the index is actually based on the name. So, this is the root node that you have and for an instance; we are taking a structure where every node can have 3 data items and 4 links and it could be it could be more it could be less, but this is just for an example. So, as you can see; so if we have this link, then on the left of Mozart, then it means all keys which are less than Mozart will be available on this link below; the link that exists here is for all keys which are greater than Mozart and less than right. Now there is nothing.

So, those will occur here. So, as you can see that Einstein, Gold, Brandt all these will come on this length Srinivasan, Singh, Wu all this come on this side the Mozart itself comes on this side. Now, if I look at this node the next level loads. Now this link has values which are less than Einstein as you can see this has values which are between Einstein and Gold. So, Einstein and I set these are values which are more than gold.

So, this is this is a and as you can see that though all nodes are shown to be of the same type as we had mentioned at the end of the 2-3-4 tree discussion, but it has variable number of entries. So, the number of links are between $n/2$ and n . So, n here is 4. So, you have at least either at least two entries or maximum up to 4 entries that can go on here.

(Refer Slide Time: 05:08)

The slide has a title 'B+-Tree Index Files (Cont.)' in red. Below the title is a small sailboat icon. The main content is a bulleted list of properties for a B+-tree:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
- A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
- Special cases:
 - If the root is not a leaf, it has at least 2 children.
 - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.

On the left side of the slide, there is vertical text: 'MANAKARAN NOORI, NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018'. At the bottom left is a video player showing a man speaking, with the text 'Database System Concepts - 8th Edition'. At the bottom right is a navigation bar with icons for back, forward, search, etc., and the text '©Silberschatz, Korth and Sudarshan'.

So, this is the basic observe definition of a B + tree. All paths from root two leaf are of the same length. This is again something you should observe here, because if you if you see all of these paths all of them have the same length here then the length is 2. So, that is a basic property of 2-3-4 tree generalized into B + tree.

So, each node that is not a root is a leaf level has between $n / 2$ to n children. Leaf node has $(n - 1) / 2$ to $n - 1$ value. And the if the root is not a leaf, then it has at least 2 children and if the root is a leaf there is no other nodes in the tree then it can have between 0 to $n - 1$ values which are quite obvious.

(Refer Slide Time: 05:54)

B⁺-Tree Node Structure

■ Typical node

P_1	K_1	P_2	\dots	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	---------	-----------	-----------	-------

- K_i are the search-key values
- P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).

■ The search-keys in a node are ordered

$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

(Initially assume no duplicate keys, address duplicates later)

Database System Concepts - 8th Edition

28.9

©Silberschatz, Korth and Sudarshan

So, naturally a typical node will look like this, where the pointers and key values alternate starting with a pointer P₁, then key K₁ and so, on and ending with a point at P_n. And the search keys are strictly ordered $K_1 < K_2 < K_{n-1}$ these are facts that we have seen for 2-3-4 tree.

(Refer Slide Time: 06:14)

Leaf Nodes in B⁺-Trees

Properties of a leaf node:

- For $i = 1, 2, \dots, n-1$, pointer P_i points to a file record with search-key value K_i
- If L_i, L_j are leaf nodes and $i < j$, L_i 's search-key values are less than or equal to L_j 's search-key values
- P_n points to next leaf node in search-key order

leaf node

Brandt	Califieri	Crick	→ Pointer to next leaf node
--------	-----------	-------	-----------------------------

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Patterson	Computer	92000
98345			

Database System Concepts - 8th Edition

28.10

©Silberschatz, Korth and Sudarshan

So, for a leaf node the pointer P_i points to the file record with the search key K_i and if there are two leaf nodes L_i and L_j and $i < j$, then $L_i \leq L_j$ search key values. So, this is

the basic ordering that we had seen in 2-3-4 tree, that is what is getting generalized for a non leaf node.

(Refer Slide Time: 06:40)



Non-Leaf Nodes in B⁺-Trees

■ Non leaf nodes form a multi-level sparse index on the leaf nodes. For a non-leaf node with m pointers:

- All the search-keys in the subtree to which P_1 points are less than K_1 ,
- For $2 \leq i \leq n - 1$, all the search-keys in the subtree to which P_i points have values greater than or equal to K_{i-1} and less than K_i
- All the search-keys in the subtree to which P_n points have values greater than or equal to K_{n-1}

P_1	K_1	P_2	\dots	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	---------	-----------	-----------	-------

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

Database System Concepts - 6th Edition

28.11 ©Silberschatz, Korth and Sudarshan

Similarly all search-keys in the subtree which P_1 points to are $< K_1$, then for all that P_n points to are $> K_{n-1}$. And in the other cases they are between the two consecutive key values that exist between the pointers.

(Refer Slide Time: 07:01)



Example of B⁺-tree

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

Database System Concepts - 6th Edition

28.12 ©Silberschatz, Korth and Sudarshan

```

graph TD
    Root[El Said, Mozart] --> L1a[El Said, Einstein]
    Root --> L1b[El Said, Gold, Kata, Kim]
    Root --> L1c[Mozart, Singh, Srinivasan, Wu]
    L1a --> L2a[Brandt, Callford, Crick]
    L1b --> L2b[El Said, Gold, Kata, Kim]
    L1c --> L2c[Mozart, Singh, Srinivasan, Wu]
  
```

B⁺-tree for instructor file ($n = 6$)

- Leaf nodes must have between 3 and 5 values ($\lceil (n-1)/2 \rceil$ and $n-1$, with $n = 6$)
- Non-leaf nodes other than root must have between 3 and 6 children ($\lceil n/2 \rceil$ and n with $n = 6$)
- Root must have at least 2 children

So, this is an example of a simple case which is n where $n = 6$.

(Refer Slide Time: 07:10)

Observations about B⁺-trees

- Since the inter-node connections are done by pointers, "logically" close blocks need not be "physically" close
- The non-leaf levels of the B⁺-tree form a hierarchy of sparse indices
- The B⁺-tree contains a relatively small number of levels
 - ▶ Level below root has at least $2 \cdot \lceil n/2 \rceil$ values
 - ▶ Next level has at least $2 \cdot \lceil n/2 \rceil \cdot \lceil n/2 \rceil$ values
 - ▶ ... etc.
 - If there are K search-key values in the file, the tree height is no more than $\lceil \log_{n/2}(K) \rceil$
 - thus searches can be conducted efficiently
- Insertions and deletions to the main file can be handled efficiently, as the index can be restructured in logarithmic time

CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition

28.13

©Silberschatz, Korth and Sudarshan

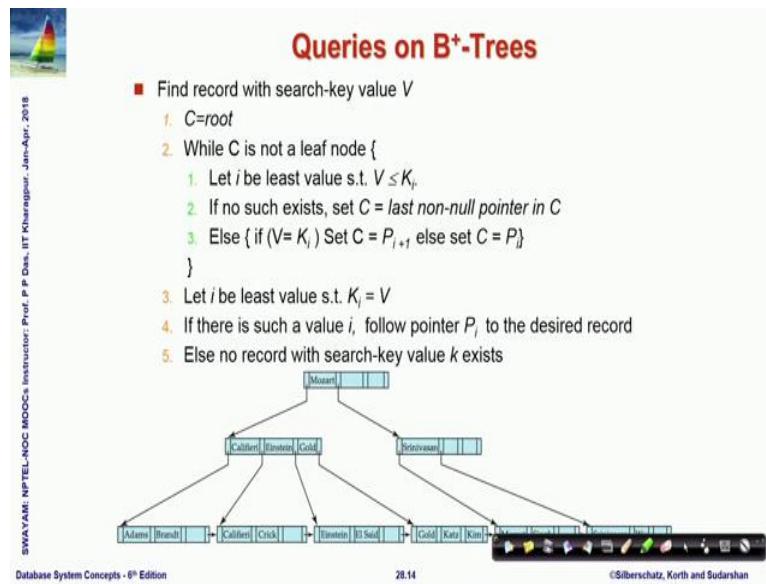
So, since the inter-node connections are done by pointers, “logically” closed blocks are not “physically” close. So, that is a key idea there is a key observation about the B + tree. So, 2 nodes the records which are logically closed are may not actually be physically close, because the pointers actually define the closeness in terms of the ordering of the values.

So, B + tree contains relatively small number of levels, we will see what that level would be? So, what will happen; if the level below root has two values at the most at least and the below that will have $n/2$ values, because every node has to be at least half full. We have said every node we will have to have $n/2$ lengths to n links it cannot be less than that less than $n / 2$ link.

So, the next level as $n / 2$, then the next level has $2 * n/2 * n/2$ and so, on. So, every time you go down you can basically increasing by a factor of $n/2$, which as you all know simply means that the number of levels or the height is $\log K$ to the base $n/2$, where K is a number of search key values that exist on the tree. So, larger the end smaller is this value. So, larger the node size is smaller is a height and therefore, the number of insertion number of you know access operations that need to be performed.

So, insertion, deletions to the main file can be handled efficiently as the index can be restructured in logarithmic time as you have just seen.

(Refer Slide Time: 09:01)



The slide title is "Queries on B⁺-Trees". It features a logo of a sailboat on the left and a decorative footer bar at the bottom right. The main content is a flowchart of a search algorithm:

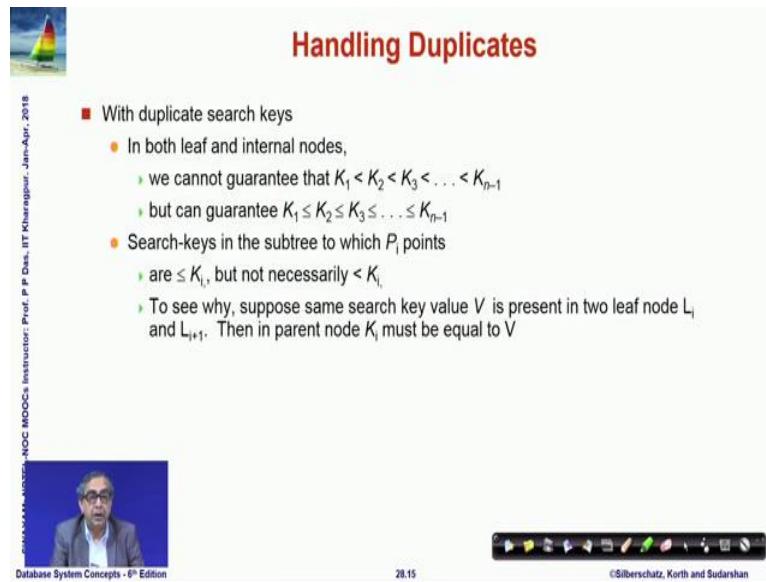
```
graph TD; C["Mozart"] --> C1["Calder"]; C --> C2["Einstein"]; C --> C3["Gold"]; C3 --> L1["Adams | Braud"]; C3 --> L2["Crick | El Said"]; C3 --> L3["Einstein | Gold | Katz | Kau | ..."]
```

Algorithm Steps:

- Find record with search-key value V
 - $C = \text{root}$
 - While C is not a leaf node {
 - Let i be least value s.t. $V \leq K_i$
 - If no such exists, set $C = \text{last non-null pointer in } C$
 - Else { if ($V = K_i$) Set $C = P_{i+1}$ else set $C = P_i$ }
 - Let i be least value s.t. $K_i = V$
 - If there is such a value i , follow pointer P_i to the desired record
 - Else no record with search-key value k exists

So, search should be very simple, because its just an extension of what you did in 2-3-4 trees. So, algorithm is given here I will skip it, because we have already done this in detail.

(Refer Slide Time: 09:13)



The slide title is "Handling Duplicates". It features a logo of a sailboat on the left and a video player window showing a professor on the right. The main content is a list of points:

- With duplicate search keys
 - In both leaf and internal nodes,
 - we cannot guarantee that $K_1 < K_2 < K_3 < \dots < K_{n-1}$
 - but can guarantee $K_1 \leq K_2 \leq K_3 \leq \dots \leq K_{n-1}$
 - Search-keys in the subtree to which P_i points
 - are $\leq K_i$, but not necessarily $< K_i$
 - To see why, suppose same search key value V is present in two leaf node L_i and L_{i+1} . Then in parent node K_i must be equal to V

Now, what we introduced I started saying that there are no duplicates. So, the keys follow strict ordering, but the whole assumption will also hold good, if you allow the equality between the consecutive keys, but only difference is there could be multiple keys which are all equal; and if that happens then you have to use the same key value

present at the two leaf nodes and the parent will also have the same leaf node same value.

(Refer Slide Time: 09:43)



Handling Duplicates

■ We modify find procedure as follows

- traverse P_i even if $V = K_i$
- As soon as we reach a leaf node C check if C has only search key values less than V
 - if so set $C = \text{right sibling of } C$ before checking whether C contains V

■ Procedure printAll

- uses modified find procedure to find first occurrence of V
- Traverse through consecutive leaves to find all occurrences of V

** Errata note: modified find procedure missing in first printing of 8th edition

Database System Concepts - 8th Edition 28.16 ©Silberschatz, Korth and Sudarshan

So, for doing in the case of such duplicates will have to a little bit modify the procedure for doing the search and say printing all values and so, on. So, you could go through that.

(Refer Slide Time: 09:58)



Queries on B+-Trees (Cont.)

■ If there are K search-key values in the file, the height of the tree is no more than $\lceil \log_{n/2}(K) \rceil$

■ A node is generally the same size as a disk block, typically 4 kilobytes

- and n is typically around 100 (40 bytes per index entry)

■ With 1 million search key values and $n = 100$

- at most $\log_{50}(1,000,000) = 4$ nodes are accessed in a lookup

■ Contrast this with a balanced binary tree with 1 million search key values — around 20 nodes are accessed in a lookup

- above difference is significant since every node access may need a disk I/O, costing around 20 milliseconds



Database System Concepts - 8th Edition 28.17 ©Silberschatz, Korth and Sudarshan

So, if there is a key search-key values in the file, then let us see what the cost is coming to actually, then the height of the tree is not more than $\log K_n / 2$. So, if we say that the every node. So, how large would be the node. Now again I would remind you that we are

moving from 2-3-4 tree, which was a in memory data structure to a external data structure. So, our main cost is a disk access. So, what would you like to make this node size, if we make the node size too small, then there will be too many nodes and every node will have to be accessed? So, as you can see this is $\log n/2$.

So, we benefit by making n larger, larger the n this log value or the height will be less, but can I make n arbitrary large then n will not fit into one disk block. So, it would it cannot be accessed in one fetch from the disk to the memory. So, we would typically like to make it is customary to make the node as the same size as the disk block, which is typically say 4 kilobyte or 8 kilobyte like that and therefore, the if that is a size then it the n will be typically around 100, because if 4 kilobytes is a is a total space and if I assume that 40 bytes per index entry, which is very typical, then n would be about 100.

So, if I assume that my index file has actually 1 million search key values to look for, then I will need 1 million to the base 100 by 250. So, 1 million $\log 1$ million to the base 50 which is approximately 4 node accesses in a lookup table. So, that is amazingly fast if you contrast this with binary balanced binary tree which will be $\log 1$ million to the base 2; which would be about 20 nodes accesses 20 disk accesses for this lookup. So, this is the core reason that B + trees are preferred and with this if even, when you have couple of million records in a in a table you can actually manage with a very small number of node accesses for the lookup, which makes the realization of algorithms possible in the next couple of slides.

(Refer Slide Time: 12:23)

Updates on B⁺-Trees: Insertion

1. Find the leaf node in which the search-key value would appear
2. If the search-key value is already present in the leaf node
 - 1. Add record to the file
 - 2. If necessary add a pointer to the bucket
3. If the search-key value is not present, then
 - 1. Add the record to the main file (and create a bucket if necessary)
 - 2. If there is room in the leaf node, insert (key-value, pointer) pair in the leaf node
 - 3. Otherwise, split the node (along with the new (key-value, pointer) entry) as discussed in the next slide

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur
Database System Concepts - 8th Edition

28.18 ©Silberschatz, Korth and Sudarshan

I have discussed about how to update B + trees talked about the insertion and the deletion process. I will skip them in the presentation, now because as we have discussed the process of insertion in depth in terms of the 2-3-4 tree the only difference here is that this is in a generalized framework, but follows exactly the same idea of node splitting and keeping in mind that in case of 2-3-4 tree you move from 2 to 3 and 3 to 4 node here. All that you will have to remember is you always make sure that you have every node half filled, because $n / 2$ is a minimum requirement.

So, you keep on inserting in a node till it becomes full, when it becomes full you cannot insert any more you divide it and split it into two nodes. So, that each one of them become at least half filled and that is the simple logic and rest of it you can figured out by following on the 2-3-4 tree insertion. So, this is the first algorithm.

(Refer Slide Time: 13:33)

The slide title is "Updates on B+-Trees: Insertion (Cont.)". It features a small sailboat icon in the top left and a photo of the instructor in the bottom left. The content is organized into sections:

- Splitting a leaf node:
 - take the n (search-key value, pointer) pairs (including the one being inserted) in sorted order. Place the first $\lceil n/2 \rceil$ in the original node, and the rest in a new node
 - let the new node be p , and let k be the least key value in p . Insert (k, p) in the parent of the node being split
 - If the parent is full, split it and propagate the split further up
- Splitting of nodes proceeds upwards till a node that is not full is found
 - In the worst case the root node may be split increasing the height of the tree by 1

Diagram illustrating the splitting of a leaf node:

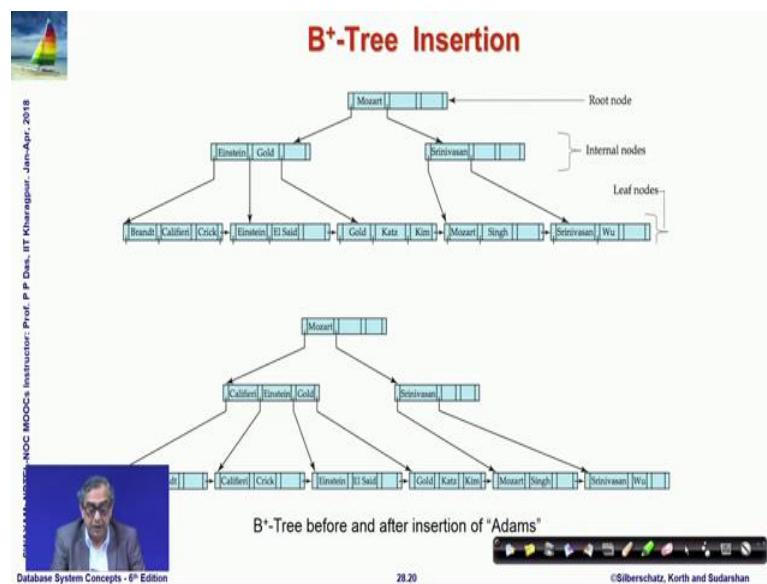
```
graph LR; A[Adams, Brandt] --> B[Califieri, Crick];
```

Result of splitting node containing Brandt, Califieri and Crick on inserting Adams
Next step: insert entry with (Califieri.pointer-to-new-node) into parent

Database System Concepts - 8th Edition 28.19 ©Silberschatz, Korth and Sudarshan

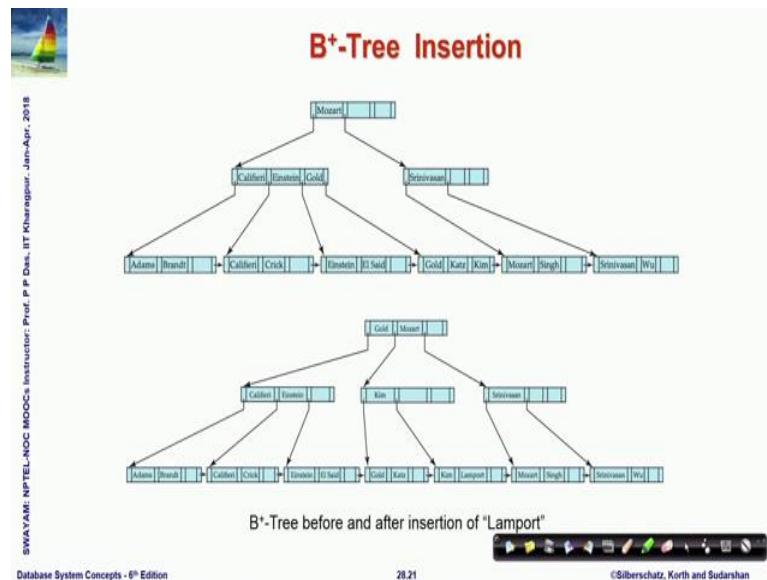
Then we have shown here the strategy to splitting the node, which I have just you know discussed and the same notion of propagating the middle element of the split continues here go to next and here the examples shown in terms of the B + tree.

(Refer Slide Time: 13:47)



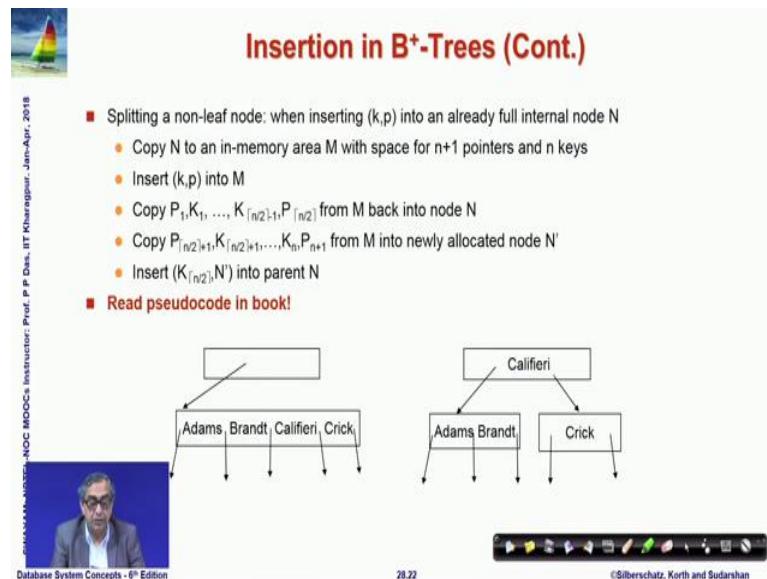
Before and after insertion of a certain key you can go through that and convince yourself.

(Refer Slide Time: 13:57)



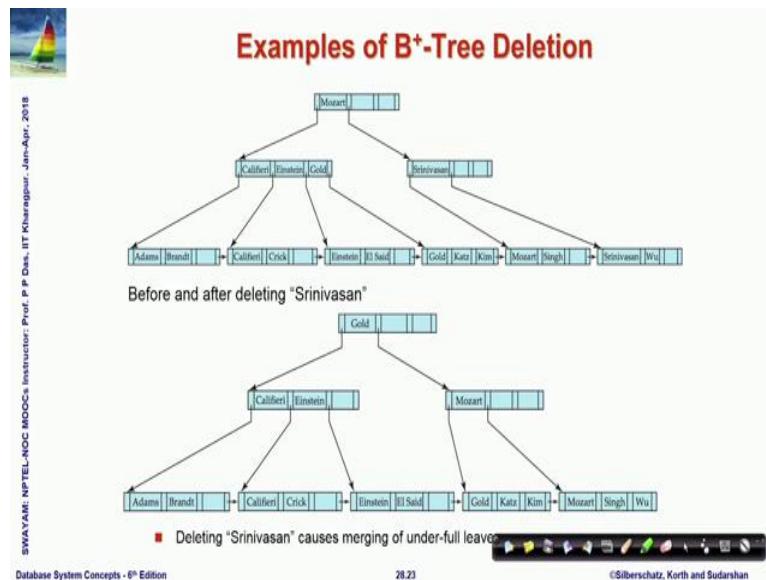
There are some more steps in the algorithm please go through them carefully and try to understand.

(Refer Slide Time: 14:04)



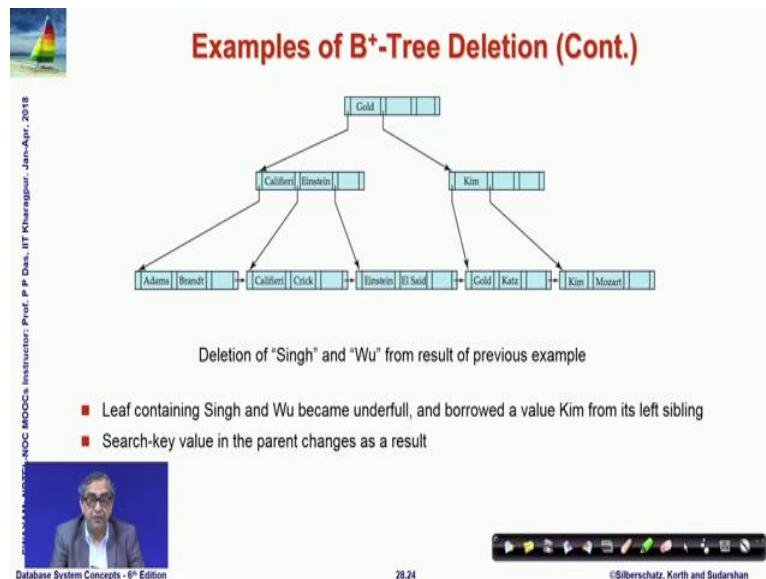
The whole process and then this is the basic algorithm written in a very cryptic pseudocode, I should say you should refer to the book actually to, for and study the whole pseudocode to understand the algorithm better and work through examples as well.

(Refer Slide Time: 14:19)



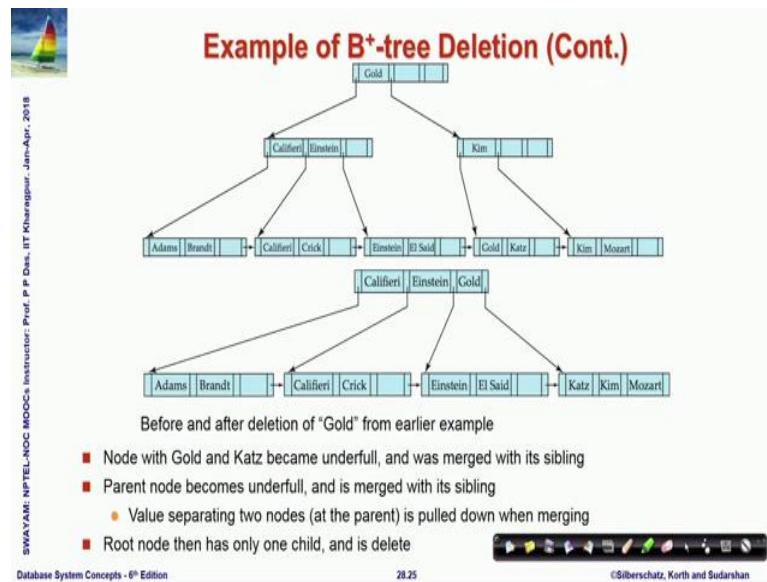
Similarly, examples of deletion in B + tree; so the trees are shown before and after deletion of Srinivasan, then if we delete like that; now in case of in contrast to splitting.

(Refer Slide Time: 14:43)



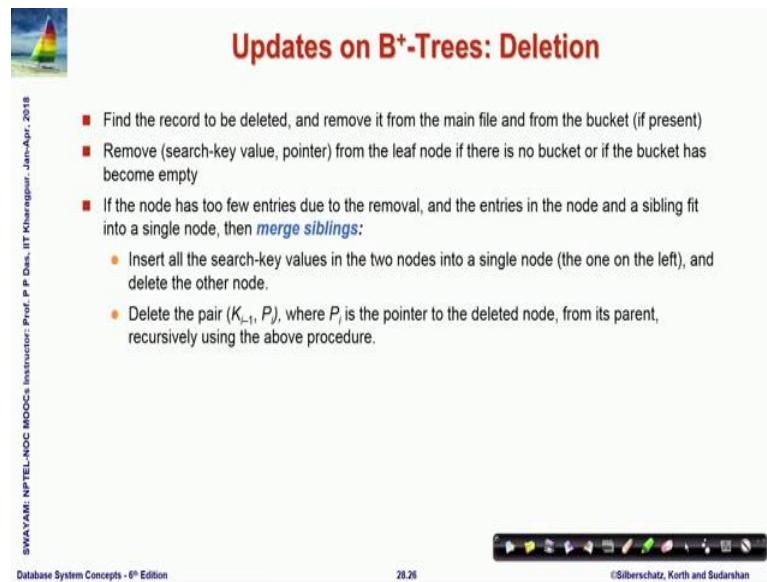
Now I will have merging of nodes which will start happening there are some more steps in the deletion shown here, please go through them and work this out they should not be you should not have any difficulty in understanding them given your background in the 2-3-4 tree.

(Refer Slide Time: 14:56)



So, more steps in the deletion. So, this is the deletion process in terms of algorithmic steps and what you need to do for deletion.

(Refer Slide Time: 15:05)



So, this is all detailed here just. So, B + tree file organization is takes care of the degradation problem.

(Refer Slide Time: 15:12)

The slide features a title 'Updates on B⁺-Trees: Deletion' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of points:

- Otherwise, if the node has too few entries due to the removal, but the entries in the node and a sibling do not fit into a single node, then **redistribute pointers**:
 - Redistribute the pointers between the node and a sibling such that both have more than the minimum number of entries
 - Update the corresponding search-key value in the parent of the node
- The node deletions may cascade upwards till a node which has $\lceil n/2 \rceil$ or more pointers is found
- If the root node has only one pointer after deletion, it is deleted and the sole child becomes the root

At the bottom left, it says 'Database System Concepts - 8th Edition'. At the bottom right, it shows '28.27' and '©Silberschatz, Korth and Sudarshan'.

In terms of the index files which would have happened, if we were used pure ordered indices like, the index sequential access method for storing the index files. So, that is now taken care of and even the data File degradation problem can also be solved by using B + Tree organization.

(Refer Slide Time: 15:20)

The slide features a title 'B⁺-Tree File Organization' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of points:

- Index file degradation problem is solved by using B⁺-Tree indices
- Data file degradation problem is solved by using B⁺-Tree File Organization
- The leaf nodes in a B⁺-tree file organization store records, instead of pointers
- Leaf nodes are still required to be half full
 - Since records are larger than pointers, the maximum number of records that can be stored in a leaf node is less than the number of pointers in a non-leaf node
- Insertion and deletion are handled in the same way as insertion and deletion of entries in a B⁺-tree index

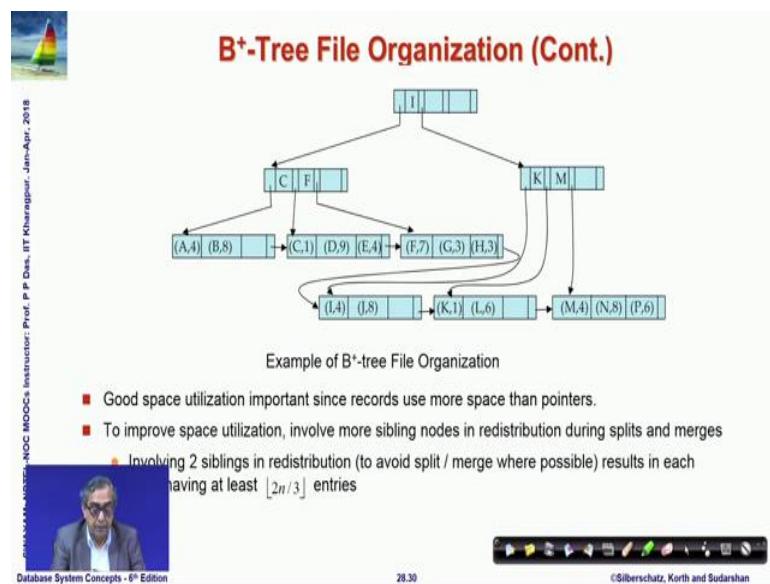
At the bottom left, it says 'Database System Concepts - 8th Edition'. At the bottom right, it shows '28.29' and '©Silberschatz, Korth and Sudarshan'.

So, it can be used for both maintaining the the index as well as the actual data file and the leaf nodes in the B + tree file organization stored the records instead of pointers. So, you finally, have the records there and the leaf nodes are still required to be half full

since they are records, but since records are larger than the maximum number of records that can be stored would be less than the number of pointers in a non leaf node insertion and deletions are handled in the same way as in the B + tree index file.

So, here all that we are explaining that. So, far we have not explained the whole B + tree in terms of index file organization and we are saying that you can do the same thing with the data file and only at the leaf level you will have to actually keep the data records for maintenance.

(Refer Slide Time: 16:47)



So, this is showing some instances of the B + tree organization.

(Refer Slide Time: 16:54)

The slide has a header 'Other Issues in Indexing' with a sailboat icon. The main content is a bullet-point list under the heading 'Record relocation and secondary indices'. The footer includes a photo of a professor, the title 'Database System Concepts - 8th Edition', the date '2018', and copyright information '©Silberschatz, Korth and Sudarshan'.

- Record relocation and secondary indices
 - If a record moves, all secondary indices that store record pointers have to be updated
 - Node splits in B+-tree file organizations become very expensive
 - Solution: use primary-index search key instead of record pointer in secondary index
 - ▶ Extra traversal of primary index to locate record
 - ▶ Higher cost for queries, but node splits are cheap
 - ▶ Add record-id if primary-index search key is non-unique

So, there is a couple of other issues the record relocation and secondary index, if a record moves all secondary indices that store record pointers will also have to be updated node splits in B + tree file organization is very expensive. So, what we do is? We use primary index search key instead of record pointer in the secondary index. So, in the secondary index we do not actually keep the direct record pointer instead, we keep the search-key of the primary index and we know that the primary index can be very efficiently searched. So, what happens is when in the secondary index when you have been able to actually find that you do not get a pointer directly to the record, but you get the search key through which you can use the primary index and actually go to that.

But with that you get yourself get rid of the requirement of maintaining different secondary index structures and getting into several record relocation problems.

(Refer Slide Time: 18:05)

The slide has a header 'Indexing Strings' with a sailboat icon. The main content lists points under two sections: 'Variable length strings as keys' and 'Prefix compression'. A small video player window shows a man speaking, and the footer includes course information and navigation icons.

- Variable length strings as keys
 - Variable fanout
 - Use space utilization as criterion for splitting, not number of pointers
- Prefix compression
 - Key values at internal nodes can be prefixes of full key
 - ▶ Keep enough characters to distinguish entries in the subtrees separated by the key value
 - E.g. "Silas" and "Silberschatz" can be separated by "Silb"
 - Keys in leaf node can be compressed by sharing common prefixes

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. De, IIT Kharagpur - Jan-Apr., 2018
Database System Concepts - 8th Edition

28.32 ©Silberschatz, Korth and Sudarshan

There are your indexing also may need to take care of other issues of string your variable length string could be keys which are variable fan out and so, the general strategy in handling indexing with string is to do a kind of what is known as prefix compression. So, you kind of find out what is the shortest prefix which can distinguish between the strings. So, if you have Silas and Silberschatz then you can easily make out that Silb would be a separating string between these two. So, Silb will match with Silberschatz, but or not will match with the first one. So, you do not need to look beyond that so, we can just keep enough characters to distinguish entries in the subtree separated I by the key values and keys in the leaf node can be compressed by sharing common prefixes.

(Refer Slide Time: 19:12)

The slide has a title 'B-Tree Index Files' at the top right. On the left is a small logo of a sailboat. The background is light blue with a grid pattern. There is some vertical text on the left edge: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Deshpande', 'Date: Jan-Apr., 2018'. The main content consists of a bulleted list and two diagrams labeled (a) and (b).

- Similar to B+tree, but B-tree allows search-key values to appear only once; eliminates redundant storage of search keys
- Search keys in non-leaf nodes appear nowhere else in the B-tree; an additional pointer field for each search key in a non-leaf node must be included
- Generalized B-tree leaf node

Diagram (a) shows a generalized B-tree leaf node structure:

P_1	K_1	P_2	\dots	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	---------	-----------	-----------	-------

(a)

Diagram (b) shows a non-leaf node structure:

P_1	B_1	K_1	P_2	B_2	K_2	\dots	P_{m-1}	B_{m-1}	K_{m-1}	P_m
-------	-------	-------	-------	-------	-------	---------	-----------	-----------	-----------	-------

(b)

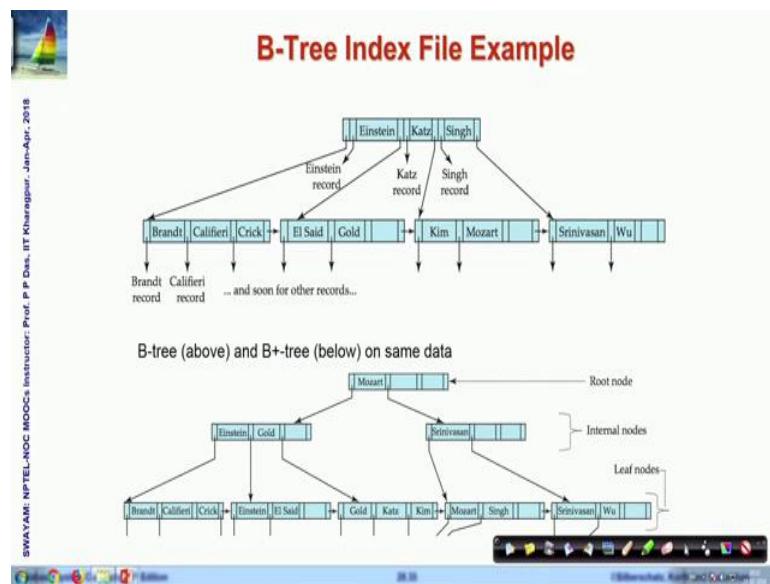
Below the diagrams is a note: ■ Non-leaf node – pointers B_i are the bucket or file record pointers.

At the bottom of the slide is a navigation bar with icons for back, forward, search, etc., and the text '28.34' and 'Bücherseiten'.

So, that is a very common strategy next let us just take a quick look into the B tree index file which is another alternate possibility the basic difference between a B + tree. And B tree allows search key values to appear only once, if you remember in the B + tree your search key values where which occurs in an internal node keeps on occurring at multiple node levels also B+, B tree does not allow that the search key non leaf nodes appear nowhere else in the B tree.

So, if it does not then naturally the question is when where will the actual record value we found out for this key. So, what you do is in the node itself you introduce another field after along with the key which is the; pointer to the actual record. So, as you can as you can see here let us get back. So, as you can see this is this was a general structure of the B + tree node. And, now what we are doing is we are putting in separate pointers along with the key which will actually maintain the data for that key which will be pointers to the actual record, because this earlier in B + tree all records. Finally, appear in terms of the leaf level nodes only they are their pointers come in the leaf level whereas, here the there is no repetition of the search key along the structure. So, they come wherever there.

(Refer Slide Time: 20:43)



So, let me just show you an example. So, if you look into this carefully. So, this is what you have seen is a B + tree. So, you can see that Mozart happened here, it also happened here and this is the leaf level. So, from here actually you get pointers to the; to the record for Mozart.

Similarly, Einstein happens here and it happens here Srinivasan happens here in. So, there are multiple times there happening this in contrast is a B tree representation where Einstein, if it happens then alongside with it the pointer to the Einstein's record exists, if brands happen here along with it the brands record exists and Einstein would not happen anywhere else in the tree. So, you do not have the second instance of the Einstein or this instance of the Mozart in the B tree. So, naturally that is the basic optimization that B tree does?

(Refer Slide Time: 21:48)

The slide has a title 'B-Tree Index Files (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under three sections: 'Advantages of B-Tree indices:', 'Disadvantages of B-Tree indices:', and 'Typically, advantages of B-Trees do not outweigh disadvantages'. The 'Advantages' section includes points about using less tree nodes than a corresponding B+ Tree and sometimes finding search-key values early. The 'Disadvantages' section includes points about larger non-leaf nodes, reduced fan-out, and more complex insertions and deletions. The footer of the slide includes the text 'CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '28.36', and '©Silberschatz, Korth and Sudarshan'.

. So, it is advantages it may use less notes than the corresponding B + tree sometimes it is possible to find the search key value even before reaching the leaf node. So, search could be efficient, but it does have a lot of disadvantages, because what happens is only small fraction of all search key values are actually found early non leaf nodes are larger.

Now, because you have pointers to the data as well, so the fan out gets reduced which means that the number of children you can have is gets reduced. So, though you are expecting to get a benefit, because you are not having to go to the leaf every time, but you pay off because your fan out gets less. So, if your fan out get less naturally the tree has a greater depth. Now, because you can you are fanning out less number of children at every node. So, it has a greater depth. So, eventually your cost increases the naturally the deletions insertions are more complicated than in B + tree and implementation is more difficult.

So, typically the advantages of B tree do not outweigh the disadvantages.

(Refer Slide Time: 23:01)

Module Summary

- Understood the design of B+-Tree Index Files in depth for database persistent store
- Familiarized with B-Tree Index Files

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

28.37

©Silberschatz, Korth and Sudarshan

So, it is not very frequent that you will use B trees, but they are used at times, but that is not a very common thing and we stick to B + tree for both of the data file as well as index file storage. So, in this module you have understood the design of B + index B + tree index files in depth for the purpose of data base persistent store and I would again remind you that whole discussion of how B + tree is organized and how operations of access insert delete are done in B + tree. I have introduced them in keeping in parallel with the simpler in memory data structure for this which is a 2-3-4 tree discussed in the last module.

So, while going through the insertion deletion processes of B+ tree, if you have difficulty following I would request that you go back to the 2-3-4 tree that is kind the simplest situation that can have that can occur and understand that and then you come back to the specific points in the B + tree algorithm and also always keep in mind. When you refer to 2-3-4 tree for understanding also always keep in mind that in case of B + tree all node types are same and the basic requirement is every node must be at least half full all the time except of course, for the root and in addition we have also familiarized with B tree and reason that B tree possibly is not a very powerful is not powerful enough it does not give enough advantages so, that to we would like to use it in place of B + tree.

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

Lecture - 29
Indexing and Hashing/4 : Hashing

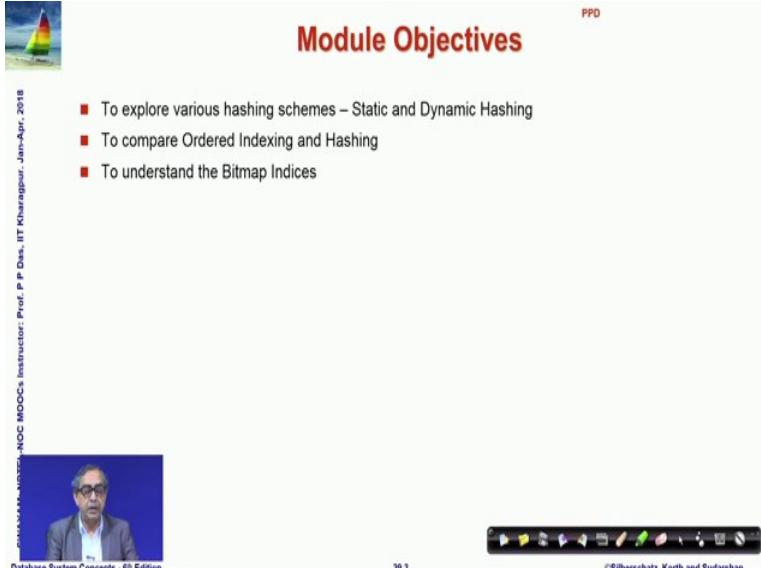
Welcome to module 29 of Database Management Systems; we have been talking about indexing and hashing and this is a fourth in the series.

(Refer Slide Time: 00:26)

The slide has a title 'Module Recap' in red at the top right. On the left, there is a small image of a sailboat on water. A vertical column of text on the far left reads: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018'. At the bottom left, it says 'Database System Concepts - 8th Edition'. In the center, there is a bulleted list: '■ B+-Tree Index Files' and '■ B-Tree Index Files'. The bottom right corner features the copyright notice '©Silberschatz, Korth and Sudarshan' next to a decorative footer bar.

In the previous 3 we have talked about different aspects of indexing and specifically in the last module, we have introduced the most powerful data structure B+ tree for index files.

(Refer Slide Time: 00:39)

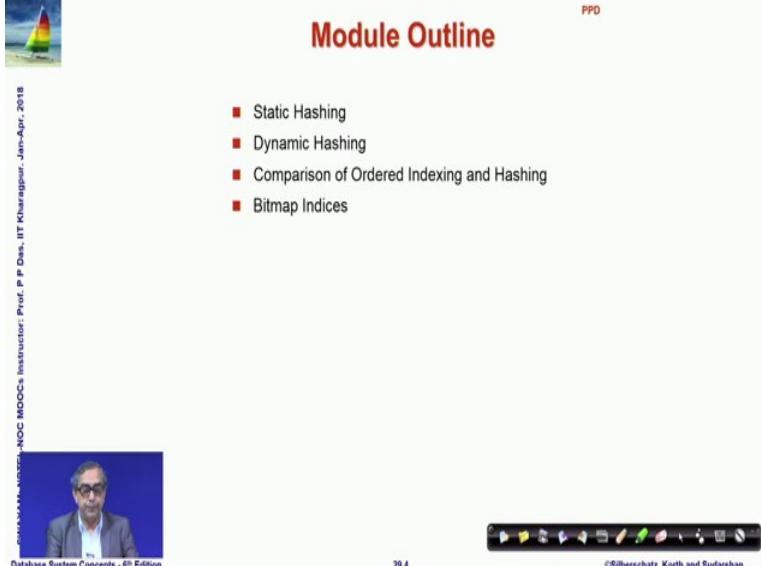


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is a video frame showing a man with glasses. The bottom right contains the text "Database System Concepts - 8th Edition", "29.3", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

- To explore various hashing schemes – Static and Dynamic Hashing
- To compare Ordered Indexing and Hashing
- To understand the Bitmap Indices

In this module, we will take a look into a explore into various hashing schemes for achieving the similar targets we will look at static and dynamic hashing. And we will then compare it between the ordered indexing that we have discussed already and hashing and we will also understand about what are called bitmap indices.

(Refer Slide Time: 01:03)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is a video frame showing a man with glasses. The bottom right contains the text "Database System Concepts - 8th Edition", "29.4", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

So, these are the module outline.

(Refer Slide Time: 01:07)

Static Hashing

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block)
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function**
- Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B
- Hash function is used to locate records for access, insertion as well as deletion
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition

29.6

©Silberschatz, Korth and Sudarshan

So, static hashing I am assuming that all of you know about basic concept of hashing. So, it what it does a there is a bucket is a unit of storage containing one or more records. So, that is the basic logical concept typically in physical terms a bucket can be a disk block. So, a hash file organization obtains we in a hash file organization; we attempt to obtain a bucket for a record directly from its search key value using a hash function.

So, this is where it becomes very different compared to the ordered indexing for which we saw all this LSM method and the B+ tree where we went through different index structure, but here we want to make use of a mathematical hash function. So, that given the key ideally I should be able to get the bucket in which that particular record containing the key exists that is the requirement.

So, hash function h is a function from the set of all search key values K to the set of all bucket addresses B . So, it is a mathematical function and it is used to locate the records for access insert as well as delete records with different search key values may be mapped to the same bucket right this is not what ideally we wanted, but it is possible there is a entire bucket has to be searched sequentially once you reach there to look at a record, we can make use of other techniques there we will come to that.

(Refer Slide Time: 02:33)

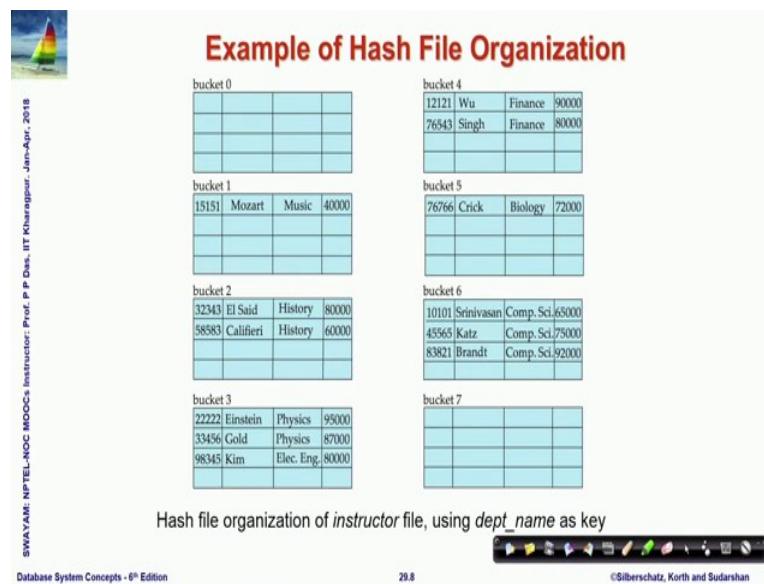
The slide has a header 'Example of Hash File Organization' with a sailboat icon. The main content discusses hash file organization for an 'instructor' file using 'dept_name' as a key. It lists four points: 1. There are 10 buckets. 2. The binary representation of the i^{th} character is assumed to be the integer i . 3. The hash function returns the sum of the binary representations of the characters modulo 10. 4. Examples include $h(\text{Music}) = 1$, $h(\text{History}) = 2$, $h(\text{Physics}) = 3$, and $h(\text{Elec. Eng.}) = 3$. The footer includes a photo of a professor, the title 'Database System Concepts - 8th Edition', page number '29.7', and copyright '©Silberschatz, Korth and Sudarshan'.

So, let us take a quick example hash file organization of an instructor file using say department named as key. So, we need to design a hash function. So, let us assume that on the address space B we have 10 buckets. So, every bucket is designated by a serial number bucket 0 to bucket 9 and we take department name is a key. So, it is a character string.

So, we take the binary representation of the i^{th} character and assume it to be the integer I simply every character you take its binary representation and think as if it is an integer. And then as a hash function we add these integer values of binary representations modulo 10. So, M hash value of music we take the binary representation of m which is the ascii code of M capital M; then add the ascii code of u the lower case u and so, on and do that modulo 10 and we get a value which is 1.

So, naturally since we are doing modulo 10 which is the number of buckets here. So, we will get a result for the hash function which is between 0 to 9 which, is a bucket address where it is expected.

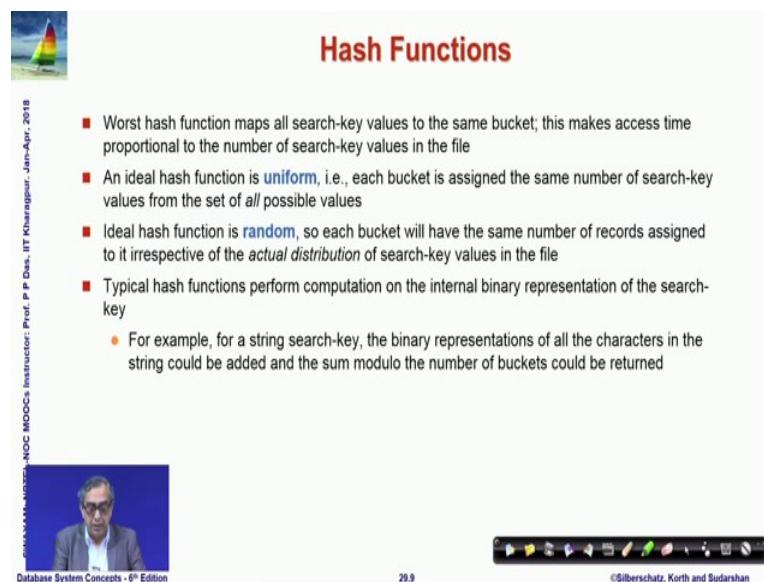
(Refer Slide Time: 03:51)



So, here we are showing an example. So, you can see in the earlier slide we are showing music is 1 history is 2 physics and electrical engineering both are hash value 3.

So, you can see here in bucket 2 since history has value 2. So, those records El Said and Califieri records go to bucket 2 whereas, physics and electrical engineering both have hash value 3. So, that Einstein gold and Kim came all go to the bucket 3 similarly it happens with the other buckets as well not all buckets are shown here shown only 8 buckets are shown, but in this way we can actually directly map them to the buckets.

(Refer Slide Time: 04:33)



And; so, such a hash function would be really useful now the question is a it is a mathematical function; so, how good or how bad it is. So, we will say that the worst possible hash function is one which maps all key values to the same bucket. So, that everything will have to within them serially; so, that is of no use.

So, the ideal one would be which will distribute the different search keys values in different buckets in an uniform manner to the from the set of all possible values. So, that would be that will be nice to have and ideal would be that if the hash function is random which means that. So, that each bucket will I mean it will generate from the key value it will generate the bucket number, it will generate the bucket address in kind of a random manner.

So, that in a random phenomena; so, that irrespective of what kind of actual distribution the search keys may have the buckets over which the distribute will be more or less the same. So, every bucket will have same number of records things will be balanced.

A typical hash function performs computation on the internal binary representation of the search key that is the basic that that is the one that you have just seen. So, if it is a string then you treat the characters as they are binary representations as integer do some modulo a number exactly what we did in the last case.

(Refer Slide Time: 05:11)

Handling of Bucket Overflows

- Bucket overflow can occur because of
 - Insufficient buckets
 - Skew in distribution of records. This can occur due to two reasons:
 - ▶ multiple records have same search-key value
 - ▶ chosen hash function produces non-uniform distribution of key values
- Although the probability of bucket overflow can be reduced, it cannot be eliminated
 - it is handled by using *overflow buckets*

Course Name: NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8th Edition

29.10

©Silberschatz, Korth and Sudarshan

Now the question is the buckets have a certain size we said that a bucket is a disk block. So, a bucket can overflow because there may not be enough sufficient buckets to keep all the records. So, it will not fit in or your distribution could be skewed. So, there may be many buckets where there are lot of space left, but some buckets may have a too many records coming on to it because of the behavior of the hash function. So, that multiple records have the same key value or chosen hash function produces non uniform distribution and so, on.

So, if that happens then the probability of bucket flow; bucket overflow will happen and we can try to reduce that, but it cannot be eliminated. So, all that you do is to have overflow bucket which is nothing, but having other buckets connected to this target bucket in a chain.

(Refer Slide Time: 07:04)

Handling of Bucket Overflows (Cont.)

■ Overflow chaining – the overflow buckets of a given bucket are chained together in a linked list

■ Above scheme is called **closed hashing**

- An alternative, called **open hashing**, which does not use overflow buckets, is not suitable for database applications

```

graph TD
    bucket0[bucket 0] --- bucket1[bucket 1]
    bucket1 --- overflow1[overflow buckets for bucket 1]
    bucket1 --- overflow2[overflow buckets for bucket 1]
    bucket2[bucket 2]
    bucket3[bucket 3]
  
```

DR. MANOHAR - NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

29.11 ©Silberschatz, Korth and Sudarshan

So, this is called a overflow chaining as you can see there are 4 buckets shown here and bucket 1 we are saying showing are connected with other two buckets which are the overflow buckets for bucket 1. So, that this kind of a scheme is called closed hashing there is an alternate scheme called open hashing, which does not use a bucket overflow and, but it is not therefore, suitable for database applications and we will not discuss it here.

(Refer Slide Time: 07:31)

Hash Indices

- Hashing can be used not only for file organization, but also for index-structure creation
- A **hash index** organizes the search keys, with their associated record pointers, into a hash file structure
- Strictly speaking, hash indices are always secondary indices
 - if the file itself is organized using hashing, a separate primary hash index on it using the same search-key is unnecessary
 - However, we use the term hash index to refer to both secondary index structures and hash organized files

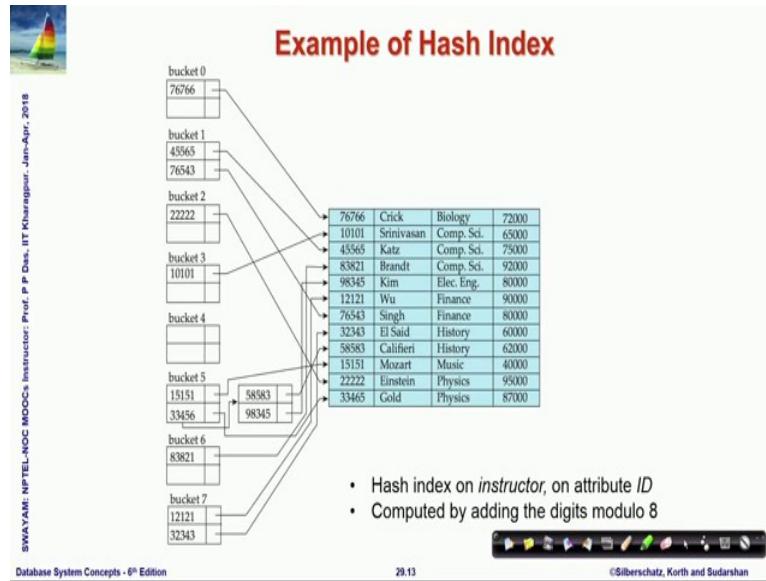
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018

Database System Concepts - 8th Edition 29.12 ©Silberschatz, Korth and Sudarshan

So, hash indices can be used only for file organization I mean not only for file organization, but they can also be used for indexed structure creation like we did for B plus tree we can use the hash indices for index structure also. So, hash index organizes the search keys with their associated record pointers into a hash file structure exactly in the same way its hashing otherwise.

So, but the you can note that the hash indices are always kind of secondary indices because if a file itself is organized using hashing; then a separate primary hash index on it using the same search keys are necessary. Because if if you are talking about primary hash indexing then it will mean that you are taking the primary search key and creating a hash index on that, but if the file is hash created by hash indexing then that already exists. So, anything that you create in terms of indexing is basically a secondary indexing structure in a hash organized file.

(Refer Slide Time: 08:38)



So, this is kind of hash indexing example. So, here I am showing the hash indexing with the ID of this table and the index is computed by adding the digits modulo 8 assuming that there are 8 buckets. So, if you take; so, if you look at bucket 0 then the key that has gone there is 76766 which is $7 + 6; 13, 20 + 6; 26 + 6 \equiv 32 \pmod{8}$ is 0.

So, it goes into bucket 0 it happens that way if, but if we look into bucket 4 you will find that the 4 IDs actually all have this value 5 under the hash function. So, they all need to go to this bucket and therefore, but the bucket size assumed here is just 2. So, after the 2 indices have gone in there a overflow chain is created and another overflow bucket is used to keep the next two IDs there. So, this is how a hash index can be created.

(Refer Slide Time: 09:45)

Deficiencies of Static Hashing

- In static hashing, function h maps search-key values to a fixed set of B of bucket addresses.
Databases grow or shrink with time
 - If initial number of buckets is too small, and file grows, performance will degrade due to too much overflows
 - If space is allocated for anticipated growth, a significant amount of space will be wasted initially (and buckets will be underfull).
 - If database shrinks, again space will be wasted
- One solution: periodic re-organization of the file with a new hash function
 - Expensive, disrupts normal operations
- *Better solution:* allow the number of buckets to be modified dynamically

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
29.14 ©Silberschatz, Korth and Sudarshan

Now, this is this kind of a scheme where you start with a fixed number of buckets and then you design a hashing function which maps the search key values to this fixed set of buckets is known as a static hashing it is static because you start with a fixed number of buckets.

So, yeah naturally the question is what should be this value of B the number of buckets. Now if it is initially too small then the file keeps on growing the performance will degrade because you will have too many overflow chains and if the all advantages of having done the hashing will get lost. On the other hand if you take a too large a B then you will unnecessarily allocate a lot of space anticipating growth, but it may take a very significant amount of time to utilize that that space or also it is possible that it the database at certain point of time grew to a large size and then it started shrinking and then again space will get wasted.

So, static hashing has these limitations. So, naturally what you will have to do is to periodically reorganize the file with a new hash function which is certainly very expensive because it changes the positions of all records. So, it disrupts the normal operation; so, it would be better if we could allow to change the number of buckets to be changed dynamically at the as the database grows. So, if the database grows it can use more and more buckets and if we could adjust this in the hashing scheme inherently; then

it will certainly be better as a solution. So, that gives rise to what is known as dynamic hashing.

(Refer Slide Time: 11:30)

The slide has a title 'Dynamic Hashing' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list under the heading 'Extendable hashing – one form of dynamic hashing'. The list includes points about generating hash values over a large range, using prefixes, and the resulting bucket address table size. A small video player window shows a person speaking, and the bottom of the slide shows a navigation bar with icons.

■ Good for database that grows and shrinks in size
■ Allows the hash function to be modified dynamically
■ Extendable hashing – one form of dynamic hashing

- Hash function generates values over a large range — typically b -bit integers, with $b = 32$
- At any time use only a prefix of the hash function to index into a table of bucket addresses
- Let the length of the prefix be i bits, $0 \leq i \leq 32$
 - ▶ Bucket address table size = 2^i . Initially $i = 0$
 - ▶ Value of i grows and shrinks as the size of the database grows and shrinks
 - Multiple entries in the bucket address table may point to a bucket (why?)
 - ▶ Thus, actual number of buckets is $< 2^i$
 - ▶ The number of buckets also changes dynamically due to coalescing and splitting of buckets

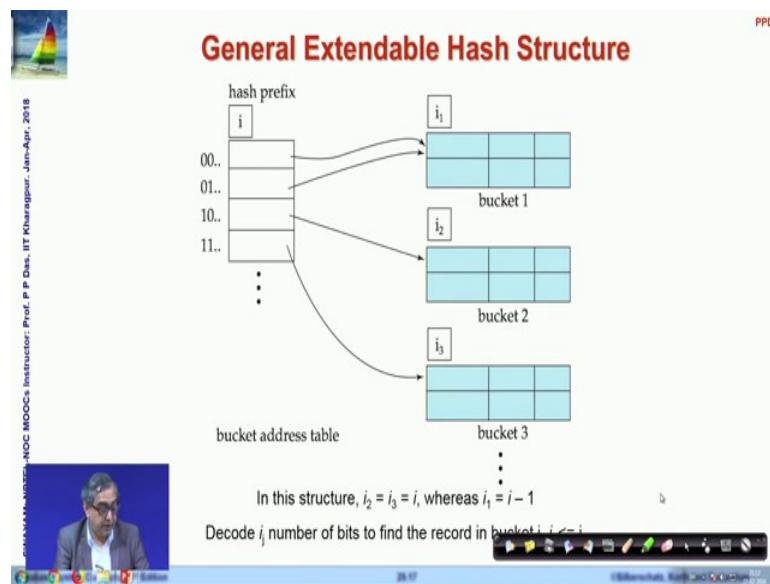
So, it is certainly good for databases that regularly grows and shrinks in size, allows the hash function to be modified dynamically. Of the different dynamic hashing schemes I will discuss the extendable hashing which is a very popular scheme. So, let us see what how it works. So, it at the hash function will generate the value over a large range say typically a B bit integer say 32 bit integer now.

So, what you have is you have generated a value which is hash value which is say over 32 bits, but what you do at any time you use only a prefix of that; you only use a part frontal part of that to index the table to the bucket address and the length of that prefix is i bits; then naturally it could be at least theoretically it could be 0 that is you do not use any prefix and it could be up to that you use all the prefixes.

And so, therefore, if you are using i bits then the bucket address table the possible you know bucket addresses that you could have is 2^i initially you keep that as 0.

So, then the address table will actually point to different buckets let us start moving to an example and see what is happening.

(Refer Slide Time: 12:57)



So, this is the general scheme. So, you have a hash prefix which is using i number of bits and therefore, different values of i number of bits. So, there will be 2^i entries naturally you have different buckets here, but you may not actually have all 2^i buckets you may have less than that as it is shown here that bucket 2 and bucket 3 exist, but bucket 1 is a holder for both this prefix 0 0 as well as prefix 0 1.

So, and on top of every bucket you have a kind of bucket depth given. So, it is a number of bits that you need to explore in the representation in the; so, that you can distinguish the different records of that bucket.

Naturally the maximum value of any of these i is the i here, but it could be less than that. So, I am sure this probably is not making much sense immediately. So, let me move to my detailed discussion.

(Refer Slide Time: 14:19)

Use of Extendable Hash Structure

- Each bucket j stores a value i_j
 - All the entries that point to the same bucket have the same values on the first i_j bits
- To locate the bucket containing search-key K_j
 - Compute $h(K_j) = X$
 - Use the first i high order bits of X as a displacement into bucket address table, and follow the pointer to appropriate bucket
- To insert a record with search-key value K_j
 - Follow same procedure as look-up and locate the bucket, say j
 - If there is room in the bucket j insert record in the bucket
 - Else the bucket must be split and insertion re-attempted (next slide)
 - ▶ Overflow buckets used instead in some cases (will see shortly)

CHAMAKAM NOETHER NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

29.18

©Silberschatz, Korth and Sudarshan

So, what I was saying that each bucket j stores a value i_j . So, this is the all the entries that point to this bucket has the same value on the first i_j bits. So, this i_j bits are identical. So, all of them have come to this bucket. So, how do you look at the bucket that contains the search key K_j (subscript)? So, it compute the hash function which is X user prefix i bits of X as a displacement into the buckets address table and follow the pointer to the appropriate bucket. $h_i(K_j)=X$

Now if I have to insert a record with a search key K_j (subscript); you will follow that same procedure as a lookup and look at the bucket j and then you will have to look for making some space. So, let me do something.

(Refer Slide Time: 15:15)

The slide title is 'Deletion in Extendable Hash Structure'. It features a small sailboat icon in the top left corner. On the right side, there is a photograph of a man with glasses and a blue background. The slide contains a bulleted list of steps for deletion:

- To delete a key value,
 - locate it in its bucket and remove it
 - The bucket itself can be removed if it becomes empty (with appropriate updates to the bucket address table)
 - Coalescing of buckets can be done (can coalesce only with a "buddy" bucket having same value of i_j and same i_{j-1} prefix, if it is present)
 - Decreasing bucket address table size is also possible
 - ▶ Note: decreasing bucket address table size is an expensive operation and should be done only if number of buckets becomes much smaller than the size of the table

Let me before going through this statement of the algorithm.

(Refer Slide Time: 15:20)

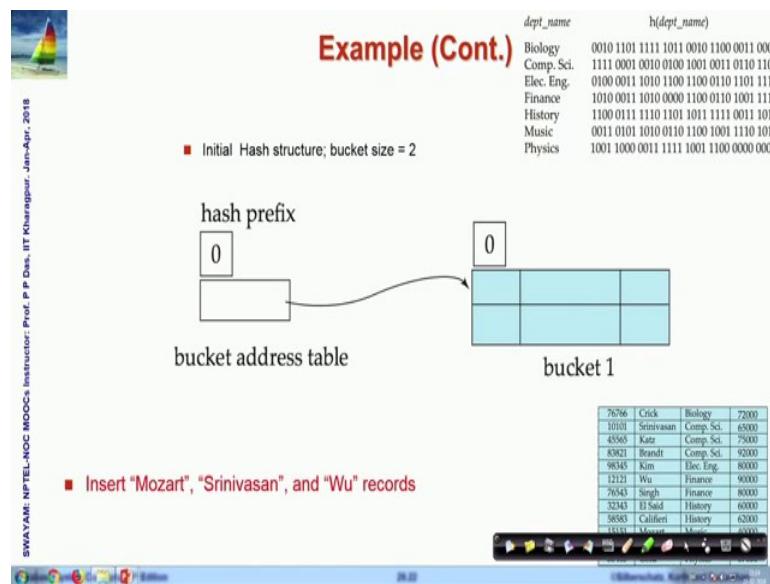
The slide title is 'Use of Extendable Hash Structure: Example'. It features a small sailboat icon in the top left corner. On the right side, there is a photograph of a man with glasses and a blue background. The slide displays a table comparing department names with their corresponding 32-bit hash values:

dept_name	$h(dept_name)$
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Let me just go through an example first and we can come back to this formal statement.

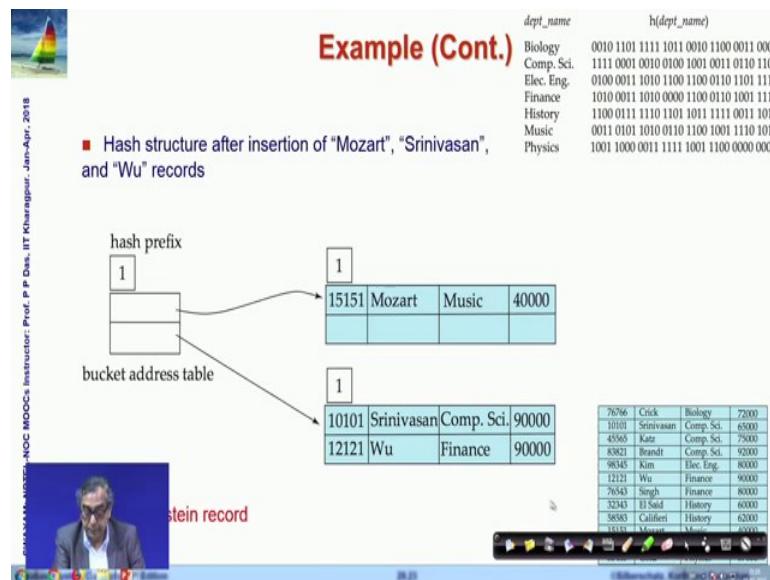
So, what we are trying to do is there is the department names which we are using as a key to do this hashing index and they are represented in terms of the binary representation. So, this is this is the hash of that department name and hashed into you can you can easily see this is 1, 2, 3, 4, 5, 6, 7, 8. So, this is hashed into 32 bit number.

(Refer Slide Time: 16:01)



Now, what do we do? So, initially we start with. So, this is all the different hash values that you can see I am sorry this is all the different hash values and this is the table that I need to actually represent. So, initially there is nothing. So, I try to I will try to insert Mozart Srinivasan and these 3 records here. So, let me try that.

(Refer Slide Time: 16:33)



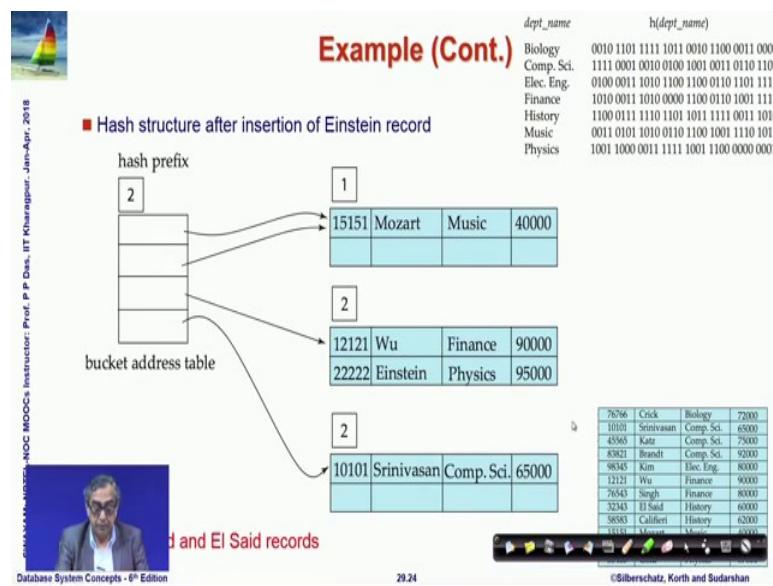
So, if I look at Mozart then Mozart is from the department of music. So, and Srinivasan is from computer science Wu is from finance. So, let us look at this. So, Mozart is from music Srinivasan is from computer science and Wu is from finance. So, these are the 3

now if we look into the prefixes of their hash values; you can see that their hash values are 1 1 and for music it is 0 right.

So, if I use a hash prefix which has just one bit and naturally therefore, I have two entries 0 and 1 then music with the value 0 maps to this bucket where I entered the record for music. And computer science and finance the records corresponding to them has a hash value prefix 1. So, they both map to discipline this is how it can get started. So, you find out while inserting you find out where is Mozart and based on that you create this.

Now, let us try to insert Einstein.

(Refer Slide Time: 18:09)



So, to insert Einstein what do we find? So, what all we already have? We have music, we have computer science, we have finance and now Einstein comes in Einstein is from physics. So, what would happen when you try to insert Einstein? You already had computer science with 1 as 1 prefix and finance with 1 prefix.

So, you had in bucket two corresponding to 1 you already have 2 records that bucket is full assuming that it can take only 2 records. So, now, you get another which is value 1; so, its value is 1. So, what I need to do? I need to actually expand this now how do I do that? I cannot expand this because there is no more space left. So, all that I need to do is to actually expand the bucket address table.

So, earlier if I just go back. So, if I just go back earlier we had only two entries because we are using only 1 bit in the prefix and with that I could not have inserted Einstein oh is from department of physics which also has a 1 bit prefix which is 1 it was. So, I need more space; so, I have increased the prefix level to 2 going here and now naturally I have increases to 2; now I have let me erase these entries and now I look at for music I look at 2 for physics 1 0, for finance 1 0, for computer science 1 1.

So, now, I find that after I have moved from looking at 1 bit of prefix to 2 bits of prefix now finance and computer science which was earlier together because I was looking at 1 bit now becomes different, but finance and physics both come to the same 1 0.

So, in the hash bucket address table 1 0 you have finance and physics coming in with Wu and Einstein records and computer science which has got 1 1 the Srinivasan record goes to a new bucket which comes from 1 1 here. Now the interesting fact is what happens to Mozart who was there if you remember the earlier structure this is we had only 1 here. So, this was going to Mozart this was 1 and Mozart was here because music had a prefix 0; now music has a prefix 0 0.

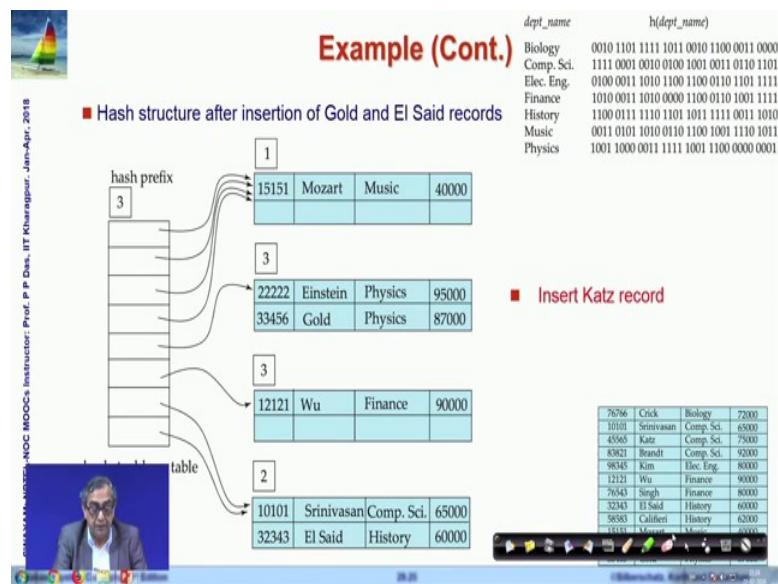
So, what he would have expected? You would have expected that therefore, since 0 0 has come and 0 1 has also come in. So, you would have expected that to have another bucket here which 0 1 is, but then you observe that actually that would be a quite a wasteful to do because you do not actually have a record which has a prefix 0 1.

Out of these two which are we are looking at two prefixes, but you do not really right now need to look at both the prefixes you can still resolve just by based on the first prefix 1. So, you do a simple trick you do not change the prefix level of the particular bucket you say it is 1. Because it is you just need to look at one bit to be able to come to the records in this bucket and the globally it has changed to 2 bits prefix, but locally you keep it as 1.

And with that what you have? You have 0 1 which has a bucket address table entry actually does not have a bucket because there is no records for that. So, you let that point to the same bucket. So, this is a very critical observation that these numbers are basically the local depth; the local information of how many bits in the prefix you need to look at to be able to resolve for coming to this bucket for the records that you currently have.

Whereas this is the global one this is a global maximum that you have. So, naturally local depth cannot exceed the global depth, but if it is equal then you have a unique mapping from the bucket address table entry to the bucket, but if the local depth as in here is less than the global depth; in terms of the number of prefix bits you are looking at then multiple bucket address table pointers actually end up in the same bucket and that is the main principle of this algorithm we can just continue further inserting gold and said into this.

(Refer Slide Time: 23:47)



So, as you try to insert gold and said gold is also from physics which we already had. So, physics and said is from history which we did not have. So, history finance computers let me let me just mark them by the side. So, you have now computer science, finance, history, music and physics.

Now, you will find that you need to you now have physics is 1 0 and you have two records for that and music is continues to be 0 0 ah; obviously, history is 1 1 same as computer science. So, that has to go on this and finances on 1 0 now, but what happens is when you try to do this; you could not have inserted more records because you have run out of space in the buckets. So, again you have run out of that; so, you need to expand in terms of the number of bits that you look at.

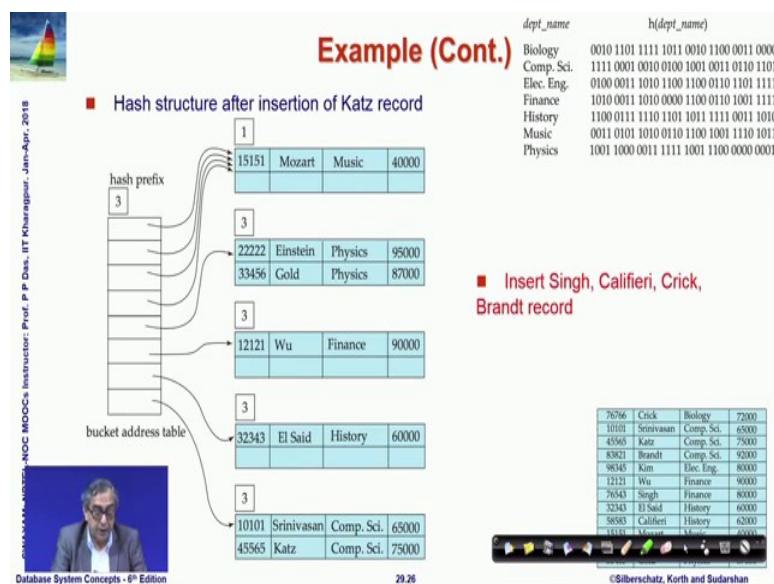
So, you increase that to 3 and now you have 0 0 0 to 1 1 1, but as I have explained not all buckets really need to look into. So, many bits Mozart this bucket continues to B with a

local depth of 1 because if you look into all these 4 different cases; then music is the only one which has a prefix 0, everyone else has a prefix 1. So, if I know that it is 0 then it comes to only this bucket and nowhere else consequently all these 4 bucket address pointers actually go to this bucket table.

Whereas these two for physics I have 1 0 and for finance we have 1 0 here and these come to. So, physics now is looking into 3; so, it is 1 0 0 finance is into 3 it is 1 0 1. So, both physics and finance go to different buckets; now coming to computer science it is 1 1 1 and there is no. So, computer science is 1 1 1 and there is no 1 1 0. So, the 1 1 0 bucket address table pointer continues to point to the same bucket and the local depth value is just 2 <3 in the global table.

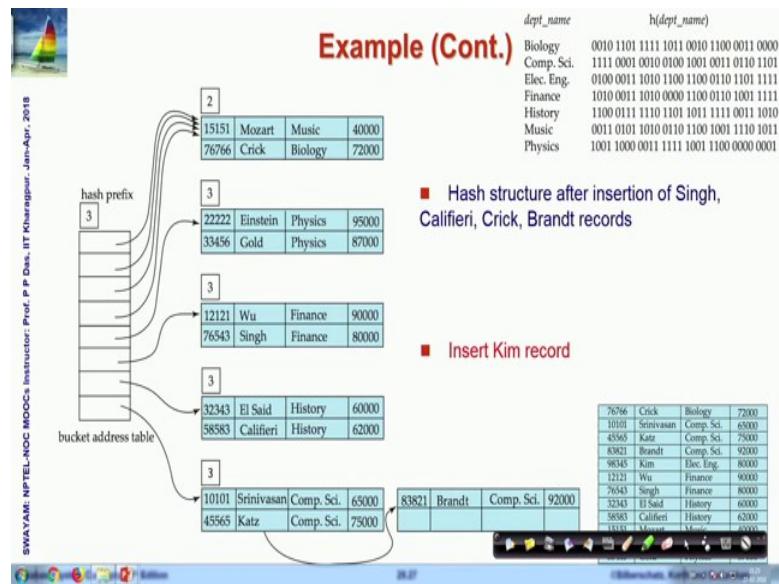
So, this is the basic process of doing dynamic hashing. So, I will not ah; so, the whole example in terms of this table I have given here worked out.

(Refer Slide Time: 26:49)



So, you can just go through every step and try to convince yourself.

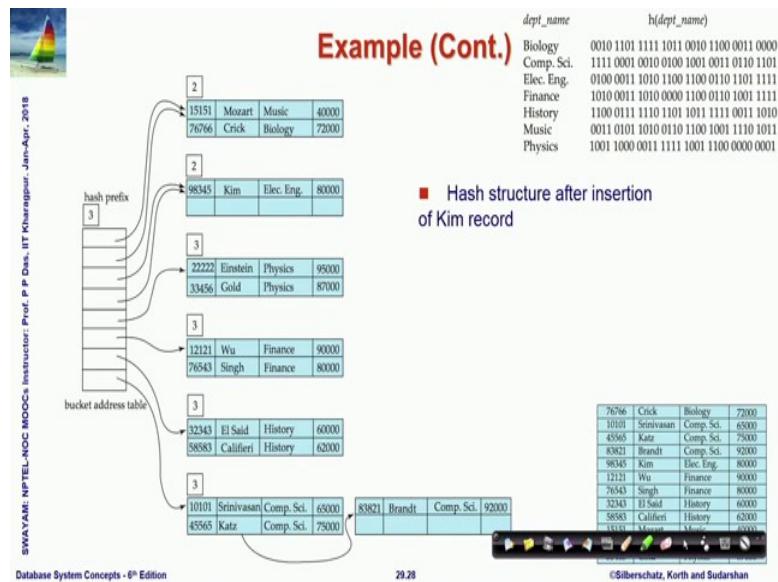
(Refer Slide Time: 26:54)



This is an interesting case that happens here where again you come to computer science professors to be entered. So, at level 3 of prefix you have all of them have prefix 1 1 1. So, you would have required to split or increase the prefix level globally the prefix level to 4, but assuming that there is an upper bound on the number of prefix levels; you can do which decides the size of the bucket address table. If that is given to be 3 you certainly cannot increase it further; so, all that you will have to do is actually do a kind of an overflow chain here as well.

So, all of them are 1 1 1 here which brings you to this you cannot find it you go to this and all 1 1 ones in future will have to be. So, it is a its kind of a tradeoff between what is the size of the global depth, how many prefixes globally you would like to look at what is the size of every bucket that you will have to maintain and what is the kind of chaining that you will have to accept because of that. So, this is what happens particularly.

(Refer Slide Time: 28:05)



So, you can continue in this way and this is a final table where all things have been hashed well. So, this is the basic extendable hashing scheme it has in this the performance does not degrade with the growth of the file and there is very minimal overhead of the space. But it does have disadvantages for example, there is a extra level of indirection to find the desired record because it the hash then come to the hash bucket address table and then go to the bucket address table itself may be very big because it is exponential in the size of the number of beds.

So, it could be larger than memory if that. So, much of you know a contiguous allocation may not be possible. So, you will need to have another possibly a B + tree structure to locate the desired record in the bucket address table first. And then changing the bucket address table will become a quite an expensive operation. So, the growth will become.

(Refer Slide Time: 28:13)

Extendable Hashing vs. Other Schemes

- Benefits of extendable hashing:
 - Hash performance does not degrade with growth of file
 - Minimal space overhead
- Disadvantages of extendable hashing
 - Extra level of indirection to find desired record
 - Bucket address table may itself become very big (larger than memory)
 - Cannot allocate very large contiguous areas on disk either
 - Solution: B*-tree structure to locate desired record in bucket address table
 - Changing size of bucket address table is an expensive operation
- Linear hashing is an alternative mechanism
 - Allows incremental growth of its directory (equivalent to bucket address table)
 - At the cost of more bucket overflows

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition 29.29 ©Silberschatz, Korth and Sudarshan

So, there are several disadvantages that also this scheme has. So, another alternate is to use a linear hashing allows incremental growth of his directory at the cost of more bucket overflows of course,.

(Refer Slide Time: 29:25)

COMPARATIVE SCHEMES

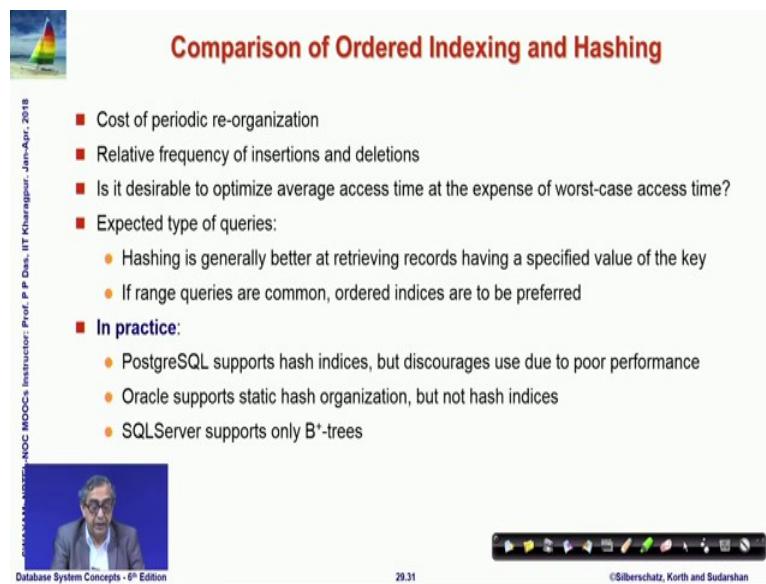
- PPD
- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition 29.30 ©Silberschatz, Korth and Sudarshan

I would quickly try to compare the two major schemes that we have discussed.

(Refer Slide Time: 29:30)



The slide title is "Comparison of Ordered Indexing and Hashing". It features a small sailboat icon in the top left corner. The main content is a bulleted list comparing the two indexing methods:

- Cost of periodic re-organization
- Relative frequency of insertions and deletions
- Is it desirable to optimize average access time at the expense of worst-case access time?
- Expected type of queries:
 - Hashing is generally better at retrieving records having a specified value of the key
 - If range queries are common, ordered indices are to be preferred
- In practice:
 - PostgreSQL supports hash indices, but discourages use due to poor performance
 - Oracle supports static hash organization, but not hash indices
 - SQLServer supports only B+-trees

At the bottom left is a small video thumbnail showing a man speaking. The bottom right contains the text "Database System Concepts - 8th Edition", the page number "29.31", and the copyright notice "©Silberschatz, Korth and Sudarshan".

The ordered indexing and the hashing now naturally ordered indexing has suffers from the cost of periodic reorganization. And because the indexing will have to be maintained the hashing is better in terms of that relative you will have to look at the relative frequency of insertion deletion that decides much of the cost between going between these two schemes.

You will have to see is it desirable to optimize average access time at the expense of worst case access time. For example there could be several ways to organize; so, that your average become your worst case may be really really bad, but as long as your averages is very good you should be happy about it. So, those kind of hashing schemes should be more preferred. So, you also depends on the kind of expected type of query.

So, for example, hashing is better in terms of retrieving records which have a specific value of the key because you can directly map from that key to the bucket. And if range queries are common then as we have seen ordered indices would make it make much better sense because in terms of the ordering you can quickly get all the required records at the same physical location nearby physical location.

If you would like to understand as to what the industry practices are it is very mixed. And if you just look into 3 of the very common database systems PostgreSQL does support hash index, but recommends does not recommend it because of the poor performance oracle supports static hash organization, but not hash indices SQL server

supports only B + trees no hash index space scheme. So, of course, you can see that there as the community is mixed in terms of it is a reaction to whether its indexing or hashing, but hashing powerful at least in limited ways is a powerful technique to go with.

(Refer Slide Time: 31:35)

The slide has a header 'PPD' and a small sailboat icon. On the left, vertical text reads 'CHAKRA MOOC NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main title 'BITMAP INDICES' is in red. Below it is a video frame showing a man speaking. A navigation bar at the bottom includes icons for back, forward, search, and other controls. The footer contains 'Database System Concepts - 8th Edition', '29.32', and '©Silberschatz, Korth and Sudarshan'.

The last two that I would like to just quickly remind I mean take you through is what is known as bitmap indexes.

(Refer Slide Time: 31:43)

The slide has a header 'Bitmap Indices' and a small sailboat icon. On the left, vertical text reads 'CHAKRA MOOC NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list about bitmap indices:

- Bitmap indices are a special type of index designed for efficient querying on multiple keys
- Records in a relation are assumed to be numbered sequentially from, say, 0
 - Given a number n it must be easy to retrieve record n
 - ▶ Particularly easy if records are of fixed size
- Applicable on attributes that take on a relatively small number of distinct values
 - E.g. gender, country, state, ...
 - E.g. income-level (income broken up into a small number of levels such as 0-9999, 10000-19999, 20000-50000, 50000- infinity)
- A bitmap is simply an array of bits

Below the list is a video frame showing a man speaking. A navigation bar at the bottom includes icons for back, forward, search, and other controls. The footer contains 'Database System Concepts - 8th Edition', '29.33', and '©Silberschatz, Korth and Sudarshan'.

Bitmap indexing is a very simple idea. So, what you it is a special type of indexing which is designed for querying on multiple keys; the basic idea is that if let us assume

that all records in a relation are numbered from 0 to n and let us say that you are talking about attributes which can take very small number of distinct values.

So, bitmap indexing is not for any attribute. So, consider attributes such a very small number of distinct value say gender which has two possible values or few possible values the country state. So, take those or maybe you can you can nominally bucket a range of numbers source income level 5, 6, 10 income levels. So, small range of possibilities and bitmap is simply array of bits. So, take an array of possible array for the records and for the possible values you mark 1 or 2 0.

(Refer Slide Time: 32:39)

Bitmap Indices (Cont.)

- In its simplest form a bitmap index on an attribute has a bitmap for each value of the attribute
 - Bitmap has as many bits as records
 - In a bitmap for value v, the bit for a record is 1 if the record has the value v for the attribute, and is 0 otherwise

record number	ID	gender	income_level
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for gender Bitmaps for income_level

m	10010	L1	10100
f	01101	L2	01000
		L3	00001
		L4	00010
		L5	00000

Database System Concepts - 8th Edition 29.34 ©Silberschatz, Korth and Sudarshan

So, this here is an example showing it. So, we are showing bitmap index for gender. So, you have a array for m the male gender and f female gender and if you look into the record 076766 has male under m gender m. And therefore, in the male gender bitmap index the first bit is 1 in f it is 0; so, actually m and f are complimentary.

Similarly, for the income levels you have 5 different bitmaps encoding; the 5 different possible levels in the income that you can have.

(Refer Slide Time: 33:21)

The slide has a title 'Bitmap Indices (Cont.)' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list:

- Bitmap indices are useful for queries on multiple attributes
 - not particularly useful for single attribute queries
- Queries are answered using bitmap operations
 - Intersection (and)
 - Union (or)
 - Complementation (not)
- Each operation takes two bitmaps of the same size and applies the operation on corresponding bits to get the result bitmap
 - E.g. $100110 \text{ AND } 110011 = 100010$
 - $100110 \text{ OR } 110011 = 110111$
 - $\text{NOT } 100110 = 011001$
 - Males with income level L1: $100110 \text{ AND } 10100 = 10000$
 - Can then retrieve required tuples
 - Counting number of matching tuples is even faster

At the bottom, there is footer text: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur', 'Database System Concepts - 8th Edition', '29.35', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative footer bar with various icons.

Now the big advantage of bitmap indices are doing different queries on multiple attributes. And for example, the often queries have intersection union and they can be simply computed in terms of bitmapped operations. So, if you have two different values to be two conditions to check in terms of bitmap indices; then you can just make there and whatever satisfy.

So, say if you are looking at males at for example, here males at income level L 1, then you can you can just take the bitmap for gender and bitmap for income level and do the ending and you get that the first record has value 1.

(Refer Slide Time: 34:08)

The slide features a title 'Bitmap Indices (Cont.)' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about bitmap indices:

- Bitmap indices generally very small compared with relation size
 - E.g. if record is 100 bytes, space for a single bitmap is 1/800 of space used by relation
 - ▶ If number of distinct attribute values is 8, bitmap is only 1% of relation size
- Deletion needs to be handled properly
 - Existence bitmap to note if there is a valid record at a record location
 - Needed for complementation
 - ▶ $\text{not}(A=v) = (\text{NOT bitmap-}A\text{-}v) \text{ AND ExistenceBitmap}$
- Should keep bitmaps for all values, even null value
 - To correctly handle SQL null semantics for NOT(A=v):
 - ▶ intersect above result with (NOT bitmap- $A\text{-Null}$)

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '29.36', and '©Silberschatz, Korth and Sudarshan'.

So, that is answer and you can quickly come to that. So, bitmap indices generally very I mean naturally they are they are they are small in compared to the relation size because you are doing bitmap indexing only if the attribute can take small number of distinct values.

Of course, the deletion has to be handled properly look at this and should keep bitmap for all values even if there are null values you must keep that otherwise you will lose track of that.

(Refer Slide Time: 34:33)

The slide features a title 'Efficient Implementation of Bitmap Operations' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about efficient bitmap implementation:

- Bitmaps are packed into words; a single word and (a basic CPU instruction) computes and of 32 or 64 bits at once
 - E.g. 1-million-bit maps can be and-ed with just 31,250 instruction
- Counting number of 1s can be done fast by a trick:
 - Use each byte to index into a precomputed array of 256 elements each storing the count of 1s in the binary representation
 - ▶ Can use pairs of bytes to speed up further at a higher memory cost
 - Add up the retrieved counts
- Bitmaps can be used instead of Tuple-ID lists at leaf levels of B⁺-trees, for values that have a large number of matching records
 - Worthwhile if > 1/64 of the records have that value, assuming a tuple-id is 64 bits
 - Above technique merges benefits of bitmap and B⁺-tree indices

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '29.37', and '©Silberschatz, Korth and Sudarshan'.

And there are several efficient implementations some information I have given, but is we do not want to go in much depth here.

(Refer Slide Time: 34:45)

Module Summary

- Explored various hashing schemes – Static and Dynamic Hashing
- Compared Ordered Indexing and Hashing
- Studies the use of Bitmap Indices for fast access of columns with limited number of distinct values

Database System Concepts - 8th Edition

29.38

©Silberschatz, Korth and Sudarshan

But several compression techniques are possible in terms of bitmaps; in the next module I will talk little bit more about how to use that. In this module to summarize we have talked about various hashing schemes static and dynamic hashing, compared the order indexing with hashing and introduced the notion of bitmap indices.

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

Lecture - 30
Indexing and Hashing/5 : Index Design

Welcome to module 30 of Database Management Systems. We have been discussing about indexing and hashing and this is a concluding module on that.

(Refer Slide Time: 00:27)

Module Recap

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

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

PPD

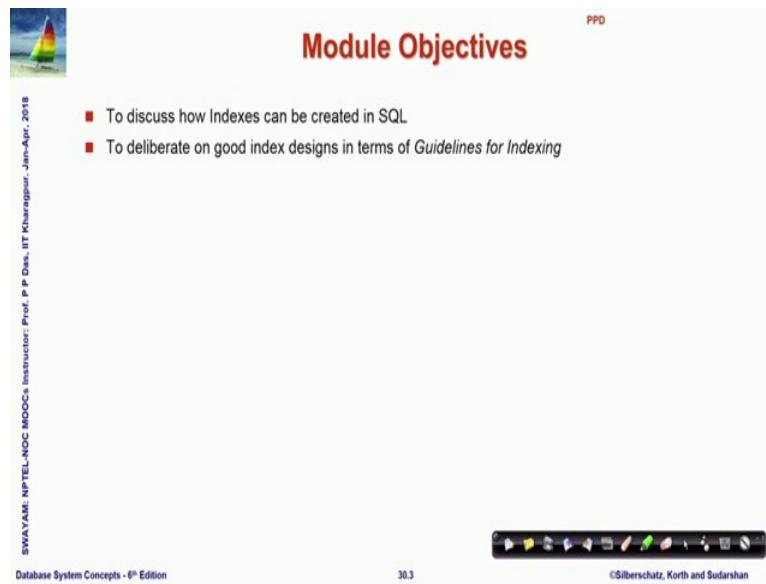
Database System Concepts - 8th Edition

30.2

©Silberschatz, Korth and Sudarshan

We have in the last module discussed about different hashing techniques static and dynamic and compare that in introduce bitmap indices.

(Refer Slide Time: 00:35)

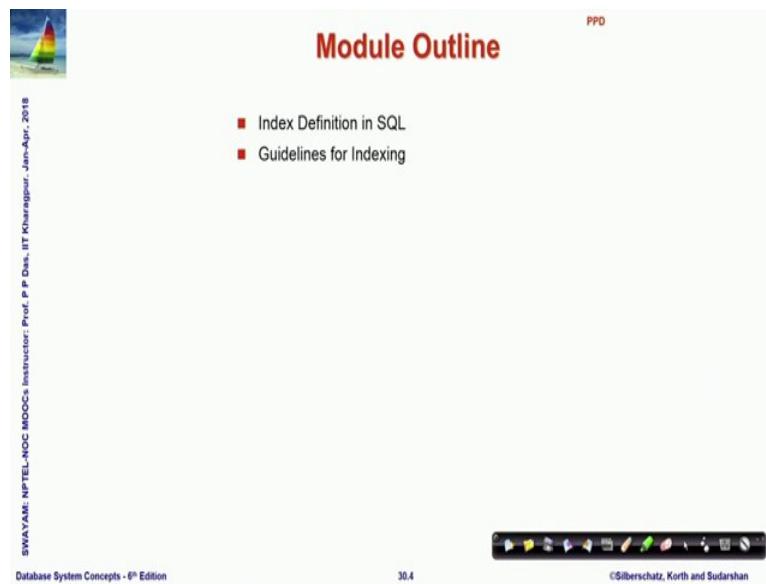


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande, IIT Kharagpur", and "Jan-Apr, 2018". At the bottom left, it says "Database System Concepts - 8th Edition". In the center, there is a bulleted list: "■ To discuss how Indexes can be created in SQL" and "■ To deliberate on good index designs in terms of *Guidelines for Indexing*". The bottom right contains the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

Now, in this module we would specifically look into the use cases, we will check as to how indexes can be created in SQL first. Because we will have to use it you have already known the theory of various different indices and so, on so, how do you actually tell the system to index.

And the second is the important thing as to when should you index and on what. So, we talked about a few guidelines for indexing.

(Refer Slide Time: 01:05)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande, IIT Kharagpur", and "Jan-Apr, 2018". At the bottom left, it says "Database System Concepts - 8th Edition". In the center, there is a bulleted list: "■ Index Definition in SQL" and "■ Guidelines for Indexing". The bottom right contains the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

So, that is that is what we want to learn.

(Refer Slide Time: 01:09)

The slide has a header 'Index Definition in SQL' and a footer with navigation icons. The content includes:

- Create an index
 - `create index <index-name> on <relation-name>`
(<attribute-list>)
 - E.g.: `create index b-index on branch(branch_name)`
- Use `create unique index` to indirectly specify and enforce the condition that the search key is a candidate key
 - Not really required if SQL `unique` integrity constraint is supported – it is preferred
- To drop an index
 - `drop index <index-name>`
 - Most database systems allow specification of type of index, and clustering
 - You can also create an index for a cluster
 - You can create a composite index on multiple columns up to a maximum of 32 columns
 - A composite index key cannot exceed roughly one-half (minus some overhead) of the available space in the data block

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018

Database System Concepts - 8th Edition 30.6 ©Silberschatz, Korth and Sudarshan

So, index can be defined in SQL in very similar syntax as you create a table. So, you say create index put a name for that index and then you say on which relation are you indexing and put the list of attributes on which you are indexing. So, if branch can relation can be indexed on branch name and I may call that b- index.

Now, there is a way to also express create unique index if you say create unique index and it will expect that the search key is a candidate key because I mean in the sense it all values of that will have to be distinct unique. Now, this used to be very common to do this kind of indexing earlier, but now it is more preferred that you can use unique integrity constraint in terms of the create table which we have already discussed. And that will ensure that you have that kind of a condition satisfied and you may not create unique index for that.

If you do not want an indexes actually do drop index and put the index name. So, most database system allow specification of the type of indexing and clustering that you want to do. So, you can create an index for a cluster also and you can create index for composite index for multiple columns.

(Refer Slide Time: 02:32)

The slide has a header 'Indexing Examples' and a footer 'Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651.htm#i1000'. It contains a bulleted list of examples:

- Create an index for a single column, to speed up queries that test that column:
 - CREATE INDEX emp_ename ON emp_tab(ename);
- Specify several storage settings explicitly for the index:
 - CREATE INDEX emp_ename ON emp_tab(ename)
TABLESPACE users STORAGE (INITIAL 20K NEXT 20k PCTINCREASE 75)
PCTFREE 0 COMPUTE STATISTICS;
- Create index on two columns, to speed up queries that test either the first column or both columns:
 - CREATE INDEX emp_ename ON emp_tab(ename, empno) COMPUTE STATISTICS;
- If a query is going to sort on the function UPPER(ENAME), an index on the ENAME column itself would not speed up this operation, and it might be slow to call the function for each result row
 - A function-based index precomputes the result of the function for each column value, speeding up queries that use the function for searching or sorting:
 - CREATE INDEX emp_upper_ename ON emp_tab(UPPER(ename)) COMPUTE STATISTICS;

So, let us run through quickly couple of examples create an index for a single column to speed up queries the test that column. So, we are saying employee emp_tab is a relation which has an attribute ename the employee name and we want to create an index on that if we do that then any search that is based on the employee name will become really fast.

Now, while you create the index you could also use optionally various other factors which relate to particularly the storage setting you can set what is you know what is the storage you want to keep for the index how you would like to increase increment that and so, on what is the table space. And very most interestingly you could say that compute statistics now this is something which is optional, but is very useful. For example, if you are not sure as to how your data is getting distributed in different relations and how really they are queried you would not know whether an index is good or it is inappropriate.

So, it is good to actually compute that statistics in terms of the index that you want to know that by doing this index what kind of accesses have happened? So, compute statistics will tell the database system to keep on computing this which you can subsequently refer to. You can create index on two columns also; so, here we are showing one where emp_tab is indexed on employee name emp_ename and employee

number together. So, you saying that you create an index on both of these and compute the statistics at the same time.

Now, other ways there are index that can be created on functions. So, suppose if there is a query which going to sort based on the uppercase writing of the e_name. So, if I just index the e_name then that itself would not speed up the operation because while you want to sort then the e_name will have to be changed into the upper case by upper and that is every time will have to do that for every record and then actually apply the sorting comparisons.

So, that will become a slow process, but you can do a function based indexing where you can specify as you can see here the function based indexing where you say that you index based on upper e_name. So, what will happen your actual values are in impossibly lower case or mixed case, but your index emp upper e_name will get created on the in the order of the upper case of e_name and will be very useful in terms of the sorting later on.

(Refer Slide Time: 05:36)

Bitmap Index in SQL

- create bitmap index <index-name> on <relation-name>(<attribute-list>)
- Example:
 - Student (Student_ID, Name, Address, Age, Gender, Semester)
 - CREATE BITMAP INDEX Idx_Gender ON Student (Gender);
 - CREATE BITMAP INDEX Idx_Semester ON Student (Semester);

STUDENT	STUDENT_ID	STUDENT_NAME	ADDRESS	AGE	GENDER	SEMESTER
Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018	100	Joseph	Alabedon Township	20	M	1
SWAYAM-NPTEL-NOCO's Instructor: Prof. P. P. Das	101	Allen	Fraser Township	22	F	1
	102	Chris	Clinton Township	20	F	2
	103	Patty	Troy	22	F	4

SEMESTER
1 1 1 0 0
2 0 0 1 0
3 0 0 0 0
4 0 0 0 1

Bitmap Index:

```

    M   1 0 0 0
    F   0 1 1 1
  
```

First Row → 2nd Row → 3rd Row → 4th Row

■ SELECT * FROM Student WHERE Gender = 'F' AND Semester = 4;
AND 0 1 1 1 with 0 0 0 1 to get the result

Source: <https://www.tutorialcup.com/dbms/bitmap-index>

Database System Concepts - 8th Edition 30.8 ©Silberschatz, Korth and Sudarshan

Now, you can like the normal index you can also create the bitmap index. So, you just say create bitmap index on the name and rest of the structure is similar. So, if there is a student relation which has these fields I can we can create an index on gender; we can create another index on semester these are very typical candidate for bitmap index

because gender can take 2 values here male and female semester can take 4 values 1, 2, 3, 4.

So, the bitmap are shown here and then if I want to do a select where the gender is F and semester is 4; then it is basically anding the bitmap of F which is 0 1 1 and the bitmap of semester 4 which is 0 0 1. So, if we if we and these two we will find that we have the result which is 0 0 0 (011 AND 001). So, which tells me that student id the fourth record of the student ID 103 is a result.

So, this is how bitmap indexing can be used in SQL.

(Refer Slide Time: 06:50)

The slide has a title 'Multiple-Key Access' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains several bullet points and a code snippet:

- Use multiple indices for certain types of queries
- Example:

```
select ID
from instructor
where dept_name = "Finance" and salary = 80000
```
- Possible strategies for processing query using indices on single attributes:
 - Use index on `dept_name` to find instructors with department name Finance; test `salary = 80000`
 - Use index on `salary` to find instructors with a salary of 80000; test `dept_name = "Finance"`
 - Use `dept_name` index to find pointers to all records pertaining to the "Finance" department. Similarly use index on `salary`. Take intersection of both sets of pointers obtained

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC Instructor: Prof. P P Doshi, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8th Edition', '30.9', and '©Silberschatz, Korth and Sudarshan'.

And actually this the whole thing can be used subsequently in multiple key access for example, if you are doing a query where it is you have department name is finance and salary is 8000, then there could be several strategies for processing this query using the index values for example, if you have single index on single attributes. So, use you can use the index on department name to find instructors which have department and finance. And then test if the salaries 80000 or you can use index on salary to find instructors with salary 80000 and then test if department name is finance.

Or you can use department name index to find pointers to all records that part in to finance department. Index on salary to find all records that part in to 80000 salary and then take intersection of the both sets to get the final result.

(Refer Slide Time: 08:04)

Indices on Multiple Keys

- Composite search keys are search keys containing more than one attribute
 - E.g. (*dept_name*, *salary*)
- Lexicographic ordering: $(a_1, a_2) < (b_1, b_2)$ if either
 - $a_1 < b_1$, or
 - $a_1 = b_1$ and $a_2 < b_2$

CHANDRAKANTAPUR NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr. 2018
Database System Concepts - 8th Edition

30.10 ©Silberschatz, Korth and Sudarshan

So, multiple key access could be achieved in terms of various single indexing single attribute indexing also; When we are doing composite search keys then naturally if there are 2 then the indexing means that you will have to define a combined lexical order. So, department name salary means that either department it is it is ordered to indexes are ordered in terms of just the department name.

Or if the department name is same then they are ordered in terms of salary this ordering in which you write the attributes in the multi composite search key is very important because you can see that for the two if the department name is same then the salary will be compared, but not the other way around.

(Refer Slide Time: 08:50)

The slide has a header 'Indices on Multiple Attributes' with a sailboat icon. The text discusses indexing combined search-keys like (dept_name, salary). It lists handling cases for WHERE clauses involving equality and less-than conditions, noting efficiency trade-offs. A video thumbnail of a professor is on the left, and navigation icons are on the right.

Suppose we have an index on combined search-key
(*dept_name*, *salary*)

- With the **where** clause
 - where *dept_name* = "Finance" **and** *salary* = 80000
the index on (*dept_name*, *salary*) can be used to fetch only records that satisfy both conditions.
 - Using separate indices is less efficient — we may fetch many records (or pointers) that satisfy only one of the conditions
 - Can also efficiently handle
 - where *dept_name* = "Finance" **and** *salary* < 80000
 - But cannot efficiently handle
 - where *dept_name* < "Finance" **and** *balance* = 80000
 - May fetch many records that satisfy the first but not the second condition

So, when you have index on multiple attributes say again going back to that same example of department naming finance and salary being 80000. So, using separate index is less efficient though we saw how that can be done, but we can also efficiently handle if we have this indexing on department name and salary. Then we can also easily handle queries like department name is finance and salary < 100; it is not because as you can easily figure out because if I can find the equality then I also know what is less.

But note that we cannot efficiently handle if I say that where department name is less than finance and balance I am sorry this should be salary is 80000. The reason is that the ordering of the attributes in this composite key is department name salary. So, if there is if department name is less than is there is no way to check for the equality of salary, but if the department name equals then there is a possibility of checking on the salary. So, because of this ordering this will may fetch many records that satisfy the first one, but not the second condition.

(Refer Slide Time: 10:16)

The slide has a header 'Privileges Required to Create an Index'. On the left, there is a small sailboat icon and a vertical text column: 'CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. On the right, there is a 'PPD' logo. The main content is a bulleted list of privileges required to create an index:

- When using indexes in an application, you might need to request that the DBA grant privileges or make changes to initialization parameters
- To create a new index
 - You must own, or have the INDEX object privilege for, the corresponding table
 - The schema that contains the index must also have a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege
 - To create an index in another user's schema, you must have the CREATE ANY INDEX system privilege
- Function-based indexes also require the QUERY_REWRITE privilege, and that the QUERY_REWRITE_ENABLED initialization parameter to be set to TRUE

Below the list is a video thumbnail showing a man speaking. The source of the slide is cited as 'Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/'.

Now, you should also remember that you need a special privilege to create an index because this is partly in the domain of the administrator's job. So, you need the specific privileges access rights to be able to do that. So, to create a new index either you have to own or own that those set of tables on which you are creating the index and or have the index object privilege for those tables or the schema that contains the index might also have a quota.

So, that you can because creating the index means you are will be using the temporary tablespace on a on a regular basis. And, but with this you will not be able to create index in some other user schema for that you need a global right which is the create any index kind of system privilege. So, also check if you are not being able to create an index check what is your privilege that exists function based indexes require other privileges; please check on that.

(Refer Slide Time: 11:28)

The slide features a small sailboat icon in the top left corner. In the top right, the text "PPD" is written above a bulleted list: "Index Definition in SQL" and "Guidelines for Indexing". The main title "GUIDELINES FOR INDEXING" is centered in large red capital letters. Below it is a video thumbnail showing a man speaking. At the bottom, there is a navigation bar with icons and the text "Database System Concepts - 8th Edition", "30.13", and "©Silberschatz, Korth and Sudarshan".

Now, let us. So, we have seen how to create index how to use that in terms of the SQL application SQL query system.

(Refer Slide Time: 11:43)

The slide has a similar layout to the previous one. It includes a sailboat icon, a bulleted list of topics, and a large title "Guidelines for Indexing". The list covers various aspects of database design and performance optimization. The video thumbnail, footer text, and navigation bar are also present.

- In Modules 16 to 20 (Week 4), we have studied various issues for a proper design of a relational database system. This focused on:
 - Normalization of Tables leading to
 - Reduction of Redundancy to minimize possibilities of Anomaly
 - Easier adherence to constraints (various dependencies)
 - Efficiency of access and update – a better normalized design often gives better performance
- The performance of a database system, however, is also significantly impacted by the way the data is physically organized and managed. These are done through:
 - Indexing and Hashing
- While normalization and design are startup time activities that are usually performed once at the beginning (and rarely changed later), the performance behavior continues to evolve as the database is used over time. Hence we need to continually:
 - Collect statistics about data (of various tables) to learn of the patterns, and
 - Adjust the indexes on the tables to optimize performance
- There is no sound theory that determines optimal performance. Rather, we take a quick look into a few common guidelines that can help you keep your database efficient.

Now, we will quickly take a look into why how should we index and where. So, if you recall in the modules 16 to 20 in the week four we have studied various issues of a proper design of relational database system, we focused on normalization of tables that can reduce redundancy and minimize anomaly how can we easily adhere to various

constraints how to improve the efficiency of access and update a better normalized design often gives better performance.

For example, we are optimizing the minimizing the requirement for computing join and all those. So, those advantages we have seen, but the actual performance of a database system is significantly impacted by the way the physical data is organized and managed which does not come across in terms of the logical design that we have seen. So, these are what are being achieved in terms of indexing and hashing. So, this is where we need to understand the actual boundary to the physical organization and that is what we have been trying to do.

So, if you think back while you are normalizing at the design level. So, those are the startup time activities; so, usually we will design and normalize and you know make the create table and do all that at the beginning of a database system. And it is really it will really be changed later because it will have severe implications, but the performance behavior will continue to evolve will continue to change because the design does not tell you exactly what the statistics of that data would be what the behavior of the data would be. So, it will evolve as data base is used over time.

Hence you will need to continuously collect statistics about the data of various tables to learn of the patterns as to which table is getting heavier which where what are the attributes on which more accesses are happening, where what kind of queries you are getting and you have to adjust the indexes on the tables to optimize the performance.

So, that is the whole requirement all about unfortunately unlike the functional dependency or multivalued dependency theories that we studied in the design space; there is no sound theory that determines optimal performance. So, all that have is more and expertise that you develop through experience. So, what I will take you through are a set of few common guidelines about how to keep your database agile while you are you go through the life cycle of different data coming in and going out.

(Refer Slide Time: 14:23)

The slide has a header 'Guidelines for Indexing' in red. On the left is a small sailboat icon. On the right is a video player interface showing a man speaking. The video player includes controls like play, pause, and volume, and displays the text 'Database System Concepts - 8th Edition' and '30.15'. The footer contains copyright information: '©Silberschatz, Korth and Sudarshan'.

■ Rule 0: Indexes lead to Access – Update Tradeoff

- Every query (access) results in a 'search' on the underlying physical data structures
 - Having specific index on search field can significantly improve performance
- Every update (insert / delete / values update) results in update of the index files – an overhead or penalty for quicker access
 - Having unnecessary indexes can cause significant degradation of performance of various operations
 - Index files may also occupy significant space on your disk and / or
 - Cause slow behavior due to memory limitations during index computations
- Use informed judgment to index!

So, the first rule I say rule 0 is the indexes lead to access update tradeoff we have already seen this at every query results in a search in the underlying physical data structure as we have understood. Having specific index on search certainly can improve performance, but as we have already noted every update with the be it insert, delete or update of values will result in update of the index files.

So, it is an overhead or penalty for quicker access that we are paying. So, having unnecessary indexes can cause significant degradation of performance. Index files will also occupy significant space on your disk and it may actually cause to slow down your behavior due to memory limitation during index computation. So, rule 0 indexes lead to this trade off always be watchful about that use judgment to index.

(Refer Slide Time: 15:26)

Guidelines for Indexing

■ Rule 1: Index the Correct Tables

- Create an index if you frequently want to **retrieve less than 15%** of the rows in a large table
 - The percentage varies greatly according to the relative speed of a table scan and how clustered the row data is about the index key
 - The faster the table scan, the lower the percentage
 - More clustered the row data, the higher the percentage
- Index columns used for joins to improve performance on **joins of multiple tables**
- Primary and unique keys automatically have indexes, but you might want to create an **index on a foreign key**
- **Small tables** do not require indexes
 - If a query is taking too long, then the table might have grown from small to large

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/

30.16 ©Silberschatz, Korth and Sudarshan

Rule 1 index the correct tables decide which tables are best candidates for index to creating an index if you frequently want to retrieve say less than 15 percent of the rows in a large table. Now first 15 percent is a ballpark number this can vary greatly according to the relative speed of the table scan ah.

Fast of the table scan you can use a lower percentage of cut off more cluster the row data you can use a higher percentage for cut up. Index tables index columns used for joining multiple tables if you have situations or multiple tables are used in joins on a on a moderately regular basis then the columns are used in the join in these tables; these tables should be indexed based on those.

The primary and unique keys automatically have indexes, but you might want to you have an index on the foreign key. So, consider that small tables do not require index if a query is requiring unnecessarily long time or unexpectedly long time; it is time to check if the table has become really big compared to small and it might be term to index that.

(Refer Slide Time: 16:45)

Guidelines for Indexing

PPD

■ Rule 2: Index the Correct Columns

- Columns with one or more of the following characteristics are candidates for indexing:
 - ▶ Values are relatively unique in the column
 - ▶ There is a wide range of values (good for regular indexes)
 - ▶ There is a small range of values (good for bitmap indexes)
 - ▶ The column contains many nulls, but queries often select all rows having a value. In this case, a comparison that matches all the non-null values, such as:
 - WHERE COL_X > -9.99 *power(10,125) is preferable to WHERE COL_X IS NOT NULL
 - This is because the first uses an index on COL_X (if COL_X is a numeric column)
- Columns with the following characteristics are less suitable for indexing:
 - ▶ There are many nulls in the column and you do not search on the non-null values
 - ▶ LONG and LONG RAW columns cannot be indexed
- The size of a single index entry cannot exceed roughly one-half (minus some overhead) of the available space in the data block

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur Date: Jan-Apr. 2018

Database System Concepts - 8th Edition

30.17

©Silberschatz, Korth and Sudarshan

So, rule 1 index the correct tables and certainly related to that is index the correct columns. The columns with some of the characteristics I have just noted down are good candidates for indexing values are relatively unique in the column, then indexing will give you a good benefit. There is a wide range of values where your regular indexes will work well.

There is a small range of values where bitmap indexes will give you good results. So, use those in column contains many nulls, but queries often select all rows having a value. So, there are column have lot of null values, but whenever you do a query then you actually take out rows which have values. So, in those cases you can actually if you involve certain I mean if you write the SQL query by keeping the condition in a way. So, that the index can be used that is for example, you could put a condition such that only non null values will be matched.

Compared to that if you have just taken (Refer Time: 17:54) at non null check your first query would run faster; if you have an index on the COL_X because the query would be able to work on that index. So, these are things that you should do in terms of the column. And if a column has the kind of characteristic that there are many nulls in the column and you typically do not search non null values; then it is good it is better not to index those columns long and long row columns cannot be indexed anyway. So, this remember the rule 2 index the correct columns.

(Refer Slide Time: 18:29)

Guidelines for Indexing

PPD

■ Rule 3: Limit the Number of Indexes for Each Table

- The more indexes, the more overhead is incurred as the table is altered
 - When rows are inserted or deleted, all indexes on the table must be updated
 - When a column is updated, all indexes on the column must be updated
- You must weigh the performance benefit of indexes for queries against the performance overhead of updates
 - If a table is primarily read-only, you might use more indexes; but, if a table is heavily updated, you might use fewer indexes

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/

30.18 ©Silberschatz, Korth and Sudarshan

Then the rule 3 limit the number of indexes for each table the more the index more overhead we have already seen this as a part of rule 2 rule 0 as well. When rows are inserted or deleted indexes of a table must be updated when columns are updated all the indexes on the column must be updated. So, there is a lot of cost; so, half as limited number of indices as will start with purposed.

So, you must regularly weigh the benefit of having the indexed for queries against the performance overhead of the updates. For example, if a table is primarily read only you might use more indexes because the overhead will be less, but if a table is heavily updated you might use fewer number of indices.

(Refer Slide Time: 19:14)

The slide has a header 'Guidelines for Indexing' in red. On the left, there's a small logo of a sailboat on water. On the right, there's a small icon labeled 'PPD'. Below the title, there's a table titled 'Table VENDOR_PARTS' with columns 'VEND ID', 'PART NO', and 'UNIT COST'. The table contains 10 rows of data. At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr-2018'. At the bottom center, it says 'Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a9651.htm'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

VEND ID	PART NO	UNIT COST
1012	10-440	25
1012	10-441	39
1012	457	4.95
1010	10-440	27
1010	65	5.10
1220	08-300	1.20
1012	08-300	1.19
1012	457	5.28

Rule 4: Choose the Order of Columns in Composite Indexes

- The order of columns in the CREATE INDEX statement can affect performance
 - Put the column expected to be used most often first in the index
 - You can create a composite index (using several columns), and the same index can be used for queries that reference all of these columns, or just some of them
- For the VENDOR_PARTS table, assume that there are 5 vendors, and each vendor has about 1000 parts. Suppose VENDOR_PARTS is commonly queried as:
 - SELECT * FROM vendor_parts WHERE part_no = 457 AND vendor_id = 1012;
 - Create a composite index with the most selective (with most values) column first
 - CREATE INDEX ind_vendor_id ON vendor_parts (part_no, vendor_id);
- Composite indexes speed up queries that use the leading portion of the index:
 - So queries with WHERE clauses using only PART_NO column also runs faster
 - With only 5 distinct values, a separate index on VENDOR_ID does not help

Rule 4 choose the order of columns in the composite index. So, you have already seen couple of slides back I talk to you to the impact of what is the impact of ordering in other columns in terms of composite index. So, the order of columns in the create index statement can affect performance. So, the column that you expected to be used most often put that as the first index.

Because it is also possible that you are actually not doing a query which takes the whole of the composite search key, but a part of it, but if you have a composite search key index you will still benefit if the query is using the attributes from the first part of the index. So, here I am showing some example say there is a vendors part table and let us say there are 5 vendors and let us say. So, there is vendor id part number and unit cost forget about unit cost for this consideration.

So, it is primarily part number and vendor id. So, let us say there are 5 vendors and each vendor has about 1000 parts. So, and let us say that it is queried like this that the part number is such and such and vendor id is such and such you get you select all all that matches.

Now, if you create a composite index then it should be on part number and vendor id not the other way around. Because if you do that then queries where only part number is used will also run faster, but the vendor id here is not a very good candidate for indexing as a first attribute because it has only five possible values very small

number of value. So, indexing really does not help here it cannot discriminate after indexing there will be lot of clusters still found.

(Refer Slide Time: 21:17)

The slide has a header 'Guidelines for Indexing' in red. On the left is a small sailboat icon. On the right is a video player showing a man speaking. The video player includes controls like play, pause, and volume, and displays the source URL: https://docs.oracle.com/cd/B10500_01/appdev/920/a965/, page 30.20, and copyright information: ©Silberschatz, Korth and Sudarshan.

Rule 5: Gather Statistics to Make Index Usage More Accurate

- The database can use indexes more effectively when it has statistical information about the tables involved in the queries
- Gather statistics when the indexes are created by including the keywords COMPUTE STATISTICS in the CREATE INDEX statement
- As data is updated and the distribution of values changes, periodically refresh the statistics by calling procedures like (in Oracle):
 - DBMS_STATS.GATHER_TABLE_STATISTICS and
 - DBMS_STATS.GATHER_SCHEMA_STATISTICS

Rule number 5 gather statistics to make index usage more accurate that is a that is a very very important factor the database can use indexes more effectively if the statistical information is available. So, gather statistics learn from that we have already discussed how to gather statistics from the create index statement.

And then there are functions these are function names in oracle in your system you might want to check up what these functions are called. So, by that you can find out statistics about the tables and the schema that you have and use that information to subsequently optimize the index.

(Refer Slide Time: 21:58)

Guidelines for Indexing

■ Rule 6: Drop Indexes That Are No Longer Required

- You might drop an index if:
 - It does not speed up queries. The table might be very small, or there might be many rows in the table but very few index entries
 - The queries in your applications do not use the index
 - The index must be dropped before being rebuilt
- When you drop an index, all extents of the index's segment are returned to the containing tablespace and become available for other objects in the tablespace
- Use the SQL command DROP INDEX to drop an index. For example, the following statement drops a specific named index:
 - `DROP INDEX Emp_ename;`
- If you drop a table, then all associated indexes are dropped
- To drop an index, the index must be contained in your schema or you must have the `DROP ANY INDEX` system privilege

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a96510/index.htm#i1000

Database System Concepts - 8th Edition 30.21 ©Silberschatz, Korth and Sudarshan

The last rule 6 is drop index that are no longer required. So, if an index might be dropped because for several reasons for example, it is simply does not speed up the queries. So, table might have become too small there will be many rows in the table, but very few index increase right we have seen these are not the ideal.

So, it may not have been the case earlier and now it may be the case. So, in that case that index should be drop because it is not helping you the queries in your application do not use the index the query you have certain indexes and the queries are done on other attributes or other composite attributes. So, it is not the indexes of no value and; obviously, index must be dropped before being rebuilt if you are rebuilding if you are creating new index in a new way then. So, make a judgment and drop indexes which are no longer required as an when you observe that in drop indexes and improve the performance.

When you drop an index all extents of the indexes segment are return to the tablespace. So, this is basically the space management and SQL command for this you already know now please keep in mind that if you drop a table then all associated indexes are automatically dropped because; obviously, if the data is not there then how about their index. So, to drop an index you need the drop any index system privilege we talked about privileges earlier too.

(Refer Slide Time: 23:26)

Module Summary

- Learned to create Indexes in SQL
- Introduced a few rules for good index

SWAYAM: NPTEL-NOC MOOCs
Instructor: Prof. P. P. Deshpande, IIT Kharagpur
Date: Jan-Apr., 2018

Database System Concepts - 8th Edition
30.22
©Silberschatz, Korth and Sudarshan

So, this summarizes our discussions on indexing and hashing. So, here in this particular module you have learned to create index in SQL and introduce few rules for good index. Overall in this week in all the 5 modules we have learnt about how to speed up query processing, how to speed up the execution of access insert delete queries in your database through the lifetime. And we have looked at various different indexing schemes, we have looked at hashing and made comparisons.

So, one take back that you can certainly have is the most important indexing scheme or indexing structure is the B + tree which can be used for data files as well as for index files and several like SQL server uses the B+ tree only. Now, hashing options we have looked at and we have seen that it has a varied acceptability it is a powerful technique, but not all systems use that equally strongly.

And we have then made understood that indexing a database or tables on different attributes is a very delicate responsibility which has to be done with a lot of judgment. And for that statistics must be rightly collected and good judgment in terms of the distribution of the data, access of the data nature of queries all these need to be considered carefully so, that you can really get good performance from the design that you have.

So, on top of the knowledge of good design that you acquired through the all the theory of normalization and you know redundancy removal; your good judgment in terms of

good appropriate index design will take you a long way in terms of making a very good database system engineer.

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

Lecture – 31
Transactions/1

Welcome to module 31 of Database Management System. This module and the few following it will be on transactions. So, we have so far, through the modules that you have done we have so far looked at the first the schema of a database system, which is the plan the layout of how the data will be organized. Then we have looked at if the data is populated, then how we can query how we can manipulate the data.

We have looked at how the data is actually physically stored, and how it can be efficiently accessed through different mechanisms in the storage. Now we will focus on the actual execution time. We will focus on what goes on when the data in a database system is accessed, it is read, locally modified and then written back. In a very simple terms this is the operation that keeps on happening in the database systems, which we will identify which we say are transactions.

(Refer Slide Time: 01:40)

Week 06 Recap

PPD

<ul style="list-style-type: none">▪ Module 26: Indexing and Hashing/1 (Indexing/1)<ul style="list-style-type: none">◦ Basic Concepts of Indexing◦ Ordered Indices▪ Module 27: Indexing and Hashing/2 (Indexing/2)<ul style="list-style-type: none">◦ Balanced Binary Search Trees◦ 2-3-4 Tree▪ Module 28: Indexing and Hashing/3 (Indexing/3)<ul style="list-style-type: none">◦ B+-Tree Index Files◦ B-Tree Index Files	<ul style="list-style-type: none">▪ Module 29: Indexing and Hashing/4 (Hashing)<ul style="list-style-type: none">◦ Static Hashing◦ Dynamic Hashing◦ Comparison of Ordered Indexing and Hashing◦ Bitmap Indices▪ Module 30: Indexing and Hashing/5 (Index Design)<ul style="list-style-type: none">◦ Index Definition in SQL◦ Guidelines for Indexing
--	---

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

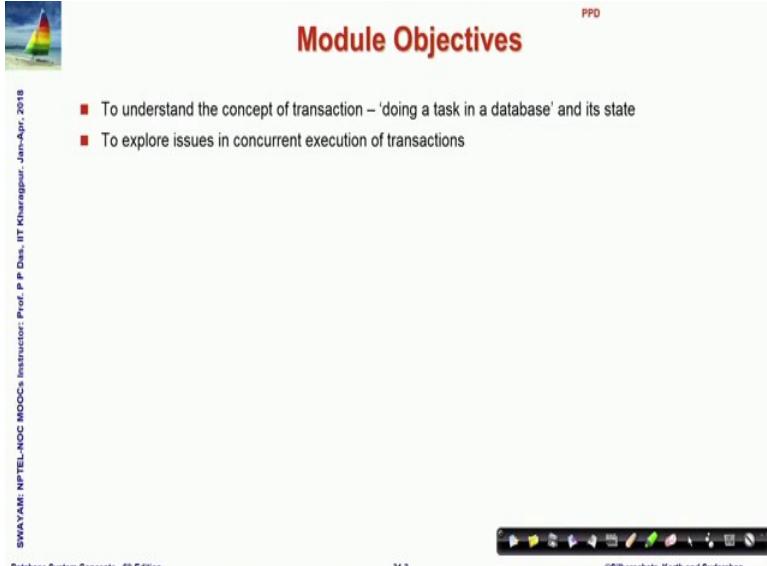
Database System Concepts - 8th Edition

31.2

©Silberschatz, Korth and Sudarshan

So, this is what we had done in the last week talking about index.

(Refer Slide Time: 01:46)



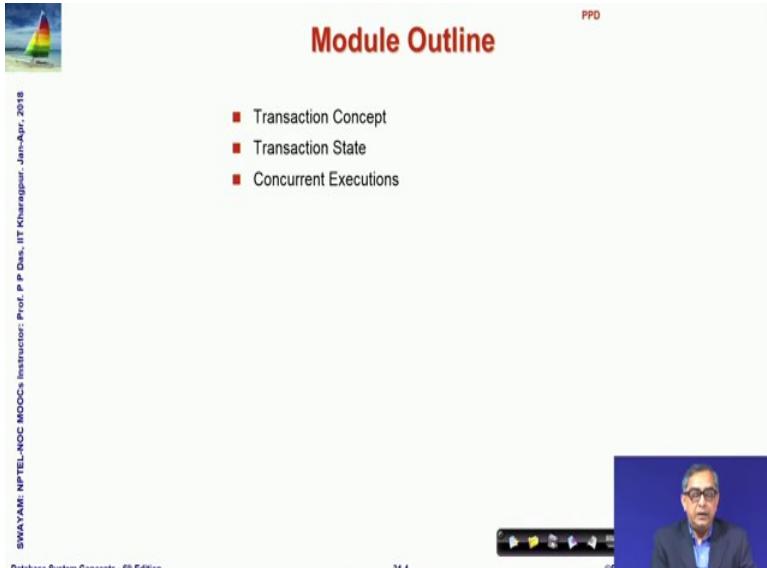
The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018". The main content area lists two objectives:

- To understand the concept of transaction – 'doing a task in a database' and its state
- To explore issues in concurrent execution of transactions

At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer includes "Database System Concepts - 8th Edition", "31.3", and "©Silberschatz, Korth and Sudarshan".

And we now start with the understanding of this concept of transaction, and we explore various issues related to concurrent execution of transactions. So, we will explain in more detail what this mean.

(Refer Slide Time: 02:03)



The slide is titled "Module Outline" in red. It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018". The main content area lists three topics:

- Transaction Concept
- Transaction State
- Concurrent Executions

At the bottom, there is a video player showing a man speaking, a navigation bar with icons, and the footer "Database System Concepts - 8th Edition", "31.4", and "©Silberschatz, Korth and Sudarshan".

And these are the 3 topics that we will focus on in this module.

(Refer Slide Time: 02:13)

The slide has a title 'Transaction Concept' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains the following text:

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items
- For example, transaction to transfer \$50 from account A to account B:
 1. `read(A)`
 2. `A := A - 50`
 3. `write(A)`
 4. `read(B)`
 5. `B := B + 50`
 6. `write(B)`
- Two main issues to deal with:
 - Failures of various kinds, such as hardware failures and system crashes
 - Concurrent execution of multiple transactions

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. At the bottom center, it says 'Database System Concepts - 8th Edition' and '31.6'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, let us first take a look at what does a transaction mean. We say transaction is a unit of program execution that accesses and possibly updates, we data items.

So, it reads possibly makes some local changes and then it writes it back. So, here is an example of a transaction. So, without that detail unnecessary details what it looks at there are two accounts account A and B, and we want to transfer 50 dollars from account A to account B. So, we want to we need to debit account A by 50 dollars, and then credit account B by 50 dollars that will achieve the transfer. So, to start this process we first need to know what is the current balance of the account A.

So, that is done by read the first instruction. Then so, A after reading a has come into a local buffer into a temporary which exists in memory, if the current balance was 200 dollar, then that has not changed that remains to be 200 dollar a has become is a local variable which is taken the value 200 dollar now. So, we locally change it we debit 50 dollars from that. So, a becomes 150 dollar, and then we write it back. So, in the account balance where it was 200 dollar, it will now become 100 and 50 dollar with this 50 dollars debited.

Then we have to do the credit process to the account B. So, in the 4th instruction we read B. So, let us say the current balance of account B was 300 dollar, then read B will make the temporary variable B as 300 we credit 300; that is, we add 50 dollars to that it

becomes 350, and then we write B onto the account based balance. So, account-based balance will now change to 350 dollar.

So, this whole sequence of 6 instructions is called a transaction. And as you can understand that to achieve our target of transferring 50 dollars from account A to account B, all these six instructions have to execute in this order so that we can get the desired results. Now the question is; so, this is pretty simple, this is like a very simple low-level program. But there are two main issues that we have to deal with. First, what is the guarantee that once the instruction one starts? What is the guarantee that it will continue up to instruction 6?

There may be some failures in between the disk may fail the hardware may fail the system may crash. So, what will happen to the state of the database? What will happen to the values that exist in the database? What will happen to the target operation that we wanted to do achieved through this transaction if such failures happen. A second issue is, we need multiple transactions to execute concurrently. What it means that, suppose I am working on a net banking updating my account I am making transfers to another party whom I need to pay. At the same time, several other people are also doing operations on their accounts, respective accounts.

Lot of other operations might happen from the database itself. For example, while I am making a transfer at that same time the database may be crediting some quarterly interest to my account. All these transactions, actually execute concurrently, which means that, they all are independently executing. They use the same CPU, but they achieve the result at the same time. So, it is not that the transactions are actually happening on separate machines, the transactions have to take effect on the same database.

So, they have to occur in a concurrent manner, that is what we see is a concurrent manner because they occur together. And while this is going on, how do we ensure, but there is one CPU. So, the CPU is executing these instructions in some order. So, how do we ensure that this in the face of such concurrency the transactions will still give me correct result? So, these are the two major issues for which we are going to study about transactions, and what we in general say the transaction management systems.

(Refer Slide Time: 07:25)

The slide has a header 'Required Properties of a Transaction' with a sailboat icon. On the left, there is a vertical sidebar with text: 'SWAYAM-NIETL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. Below the sidebar is a video player showing a man speaking. The main content area contains a section titled 'Atomicity requirement' with three bullet points. It also includes a code snippet for a transaction transfer and copyright information.

- Atomicity requirement
 - If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
 - ▶ Failure could be due to software or hardware
 - The system should ensure that updates of a partially executed transaction are not reflected in the database

Transaction to transfer \$50 from account A to account B:

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Database System Concepts - 8th Edition 31.7 ©Silberschatz, Korth and Sudarshan

So, we first set the targets we put some required properties of a transaction. The first requirement is atomicity.

Suppose again just look at the same transaction, suppose the system crashes there is a system failure after the first three instructions has happened and 4th instruction was about to happen. So, what will happen? The already the account A has been debited by 50 dollars and account B has not yet been credited with that 50 dollars. So, simply at this point if the transaction if the system failure happens, then simply 50 dollars will disappear from the system it will not exist. So, the basic requirement is that once a transaction start it should either completely happen it should either do all the 6 instruction as in this case or it should do nothing.

So, there is an all or none kind of requirement that is what we say it is like. So, transactions in a way are indivisible or atomic and this is the atomicity requirement.

(Refer Slide Time: 08:35)

The slide has a header 'Required Properties of a Transaction' with a sailboat icon. It includes a sidebar for 'SYNOPSIS: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. The main content is a section on 'Consistency requirement' with a bulleted list and a code example for a transaction transfer.

Consistency requirement

- In example, the sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
 - Explicitly specified integrity constraints
 - primary keys and foreign keys
 - Implicit integrity constraints
 - sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction, when starting to execute, must see a consistent database
- During transaction execution the database may be temporarily inconsistent
- When the transaction completes successfully the database must be consistent
- Erroneous transaction logic can lead to inconsistency

Transaction to transfer \$50 from account A to account B:

- read(A)
- $A = A - 50$
- write(A)
- read(B)
- $B = B + 50$
- write(B)

The second requirement is called consistency requirement. That is as the transactions are making changes to the database at, through these changes the integrity of the database the consistency of the values should not get affected.

So, if we there are certain specific integrity constraints we have talked of primary keys foreign keys and so on. And there could be implicit domain integrity constraints. For example, in this accounting case if we are making transfers, then while making a transfer from account A to account B the sum of the balances in account and account B before the transfer and after the transfer must be same.

So, money should not disappear, neither should get should it get generated. So, what we assume that a transaction when it starts to execute. It must start in a consistent database which is correct in every respect. During the transaction there may be temporary inconsistency. For example, if you look at instruction 4 or instruction 5 at this time, the account A has already been debited by 50 dollars the account B has not been credited by that 50 dollar. So, if you add instruction 4 if you try to see, what is the sum of the balance in account A and account B you will see that sum is 50 dollars less. But when the transaction completes, it completes the instruction 6, then again, the sum will be same as thus as it were at the beginning.

So, at the beginning of an execution, and at the end of a successful execution the database must be consistent in between there may be transient inconsistency. So, this is called the consistency requirement.

(Refer Slide Time: 10:28)

Required Properties of a Transaction (Cont.)

Isolation requirement

- If between steps 3 and 6 (of the fund transfer transaction) , another transaction **T2** is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

T1	T2
1. <code>read(A)</code> 2. <code>A := A - 50</code> 3. <code>write(A)</code>	<code>read(A), read(B), print(A+B)</code> 4. <code>read(B)</code> 5. <code>B := B + 50</code> 6. <code>write(B)</code>

- Isolation can be ensured trivially by running transactions **serially**
 - That is, one after the other
- However, executing multiple transactions concurrently has significant benefits

SWAYAM: NPTEL-NOC/NOCCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
31.9 ©Silberschatz, Korth and Sudarshan

The third is again first look at the example the same on the left is a transaction T1 which is the transaction we have been talking of. And suppose there is another transaction T2 which happens concurrently. If it happens concurrently the transaction T2 has let us say 3 instructions read (A) read (B) and print (A + B). So, it tries to read the balance of account A and B and prints their sum.

Obviously, if the transaction T2 is allowed these 3 instructions of transaction T2 if they are allowed to be executed; between instruction 3 and instruction 4 of transaction T1, then T2 will print a sum of $A + B$ which is , than the $A + B - 50$ at the beginning. So, it will become it will appear as if there is some inconsistency that has happened.

So, the isolation requirement says that when transactions occur concurrently, the net effect of the transactions should be as if they happen either first T1 happened and then T2 happen. Over first u or T2 executed and then T1 executed. The though even though they can may execute in a concurrent or mixed manner, the result of such inconsistent state of the database should not be available to the other transactions. So, this is called the isolation requirement.

So, that transactions need to be isolated appropriately; so that they can obviously if they execute serially then the isolation is trivially satisfied, but that will mean that your throughput, your performance will be very low. So, we need transactions to happen concurrently, but the isolation must be satisfied.

(Refer Slide Time: 12:24)

The slide has a title 'Required Properties of a Transaction' in red at the top right. To the left of the title is a small icon of a sailboat on water. On the far left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kharagpur - Jan-Apr- 2018'. Below the title, there is a section titled 'Durability requirement' with a bullet point: 'Once the user has been notified that the transaction has completed (i.e., the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures'. To the right of this text is a code block labeled 'Transaction to transfer \$50 from account A to account B:' followed by a numbered list of six steps: 1. `read(A)`, 2. `A := A - 50`, 3. `write(A)`, 4. `read(B)`, 5. `B := B + 50`, 6. `write(B)`. At the bottom of the slide are navigation icons and the text 'Database System Concepts - 8th Edition', '31.10', and '©Silberschatz, Korth and Sudarshan'.

The 4th is called the durability requirement, which says that if a transaction has finally completed successfully. Then the update the changes that the database that has been made by the transaction, that must persist even if there is some software or hardware failures in future so once.

This transaction of transferring 50 dollar from A to B has successfully completed with the six instructions having been executed and the money have been transferred, that will should persist even if subsequently some error some failures in the database will occur. So, it must be the changes must be durable.

(Refer Slide Time: 13:11)

ACID Properties

A **transaction** is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

- **Atomicity:**
 - Either all operations of the transaction are properly reflected in the database or none are
- **Consistency:**
 - Execution of a transaction in isolation preserves the consistency of the database
- **Isolation:**
 - Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions
 - That is, for every pair of transactions T_i and T_j , it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished
- **Durability:**
 - After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

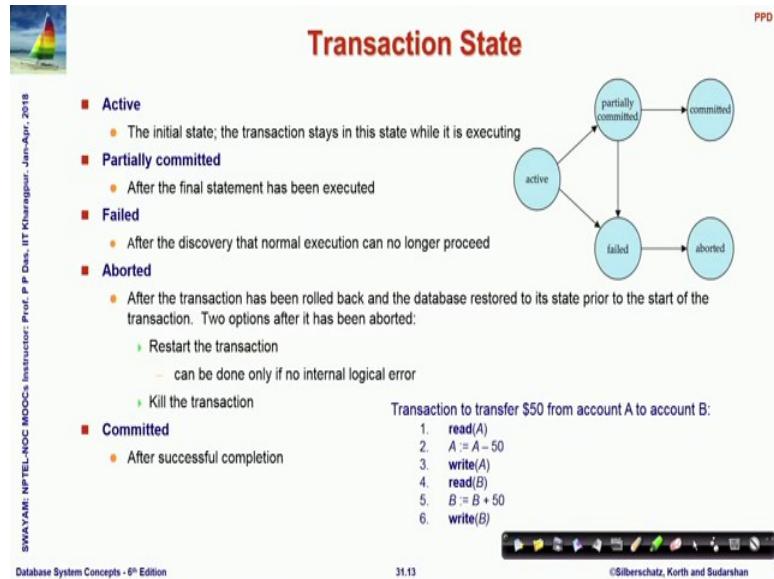
Database System Concepts - 8th Edition 31.11 ©Silberschatz, Korth and Sudarshan

So, these 4 properties are together called the acid properties of a transaction system. So, acid means a for atomicity that either all operation of the transaction are properly reflected in the database or none of them are reflected consistency c for consistency execution of a transaction in isolation preserves the consistency in the database.

The isolation requirement, that if multiple transactions are occurring concurrently, transaction T_i , T_j occurring concurrently that is some instruction of T_i happen then some instructions of T_j happen then some instruction of T_i again happen and in this manner. Even then, the final result should look like as if T_i has happened followed by T_j or T_j has first executed followed by T_i the isolation i for isolation and finally, durability once the successfully transactions have completed the changes in the database should persist. So, A C I D the acid properties are the critical properties of the transaction system and must always be satisfied.

Next what we look at is as transactions go through each and every instruction.

(Refer Slide Time: 14:29)



The transaction happened to be in one of the different states. So, while the transaction as soon as the transaction starts and starts executing starting from the initial state it is in an active state. So, consider this same transaction is done read it is in active state it has decremented A by 50 it is in active state and so on. So, as long as it is executing, it is in active state, unless it has first let us talk about success.

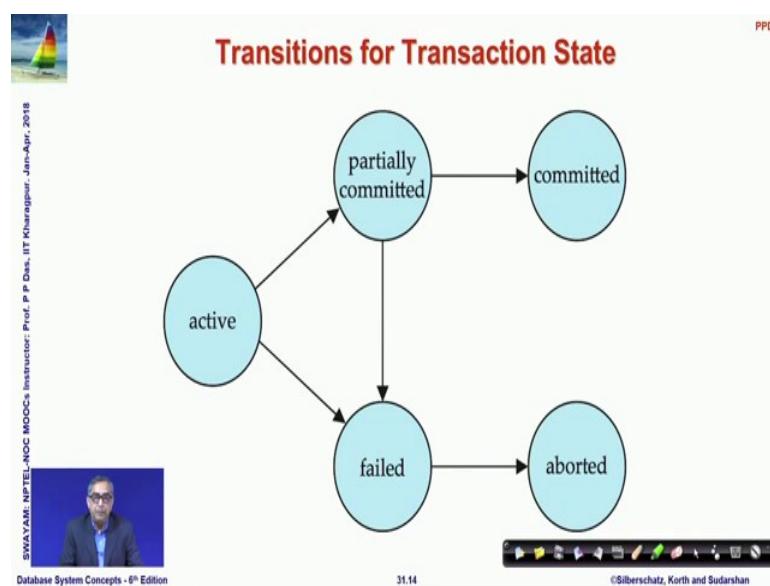
So, once it has executed the last treatment, last instruction that is instruction 6 here, it is in a state that is called partially committed. So, it has been able to successfully complete all the instructions. Or it might happen that during being in the active state, or being in the partially committed state some errors has happened so that the normal execution cannot proceed any further. Then, the transaction comes in to the failed state. A transaction which is in the failed state will eventually get aborted, because it is not known when the failure has happened.

So, naturally at the time of failure there could be an inconsistency failure could have happened in the 4th instruction in this transaction and as you have already noted. That A has already been debited by 50 dollars and B has not been credited that 50 dollars so, it is in an inconsistent state. Say failure if the transaction is in a failed state like this then we need to rollback, we need to undo the changes that we have done. We need to credit back the 50 dollars that was debited from A, so that we can reach a consistent state.

And once we have done that once we have done this rollback successfully, the transaction goes to an aborted state that is it could not take place, and after that you have 2 choices either you can restart the transaction, or you can totally kill the transaction do not do it at all depending on different situation that choice is made. In the other case if it is it were partially committed, then all instructions had completed, now the bookkeeping and other actions were required. If there is some failure during that time from partially committed it comes to fail state, and then goes to abort state as I have already explained. Or it actually commits all the changes correctly and it has completed successfully and it goes to a committed state where the transaction has successfully finished.

So, every transaction will go through this state at any point of time a transaction will be in one of the states, and depending on the status of execution it will continue to remain in that state or will change state.

(Refer Slide Time: 17:26)



So, this state transition diagram for transactions are very important, and you must thoroughly understand what is happening and remember this particular state transition ok. Now let us look into the actual concrete execution situations.

(Refer Slide Time: 17:44)

The slide has a header 'Concurrent Executions' with a sailboat icon. The main content lists advantages of concurrent executions and describes concurrency control schemes. A video player window shows a speaker, and the footer includes course information and navigation icons.

Advantages of Concurrent Executions:

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
 - Increased processor and disk utilization**, leading to better transaction *throughput*
 - For example, one transaction can be using the CPU while another is reading from or writing to the disk
 - Reduced average response time** for transactions; short transactions need not wait behind long ones

Concurrency control schemes – mechanisms to achieve isolation

- That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
31.16 ©Silberschatz, Korth and Sudarshan

So, in the concurrent execution situation, what we have we have multiple transactions that run at the same time on the system. So, that will advantages it will increase throughput, it will increase processor and disk realization for example, when one transaction is doing some operations with on the CPU, some internal computations are going on the disk can still be accessed by another transaction to read or write some values.

So, the throughput will increase and also the average response time will reduce because there may be a short transaction which if it were serially done then it will have to wait for a very long transaction, which may already been executed executing, but if we allow concurrent execution then in between that long transaction few cycles may be taken to execute the short transaction and the average response time will improve.

So, that is our basic requirement. Naturally we need to do this in a controlled manner so that we ensure that the acid property is the consistency of the database and the acid properties are maintained.

(Refer Slide Time: 18:50)

The slide has a header 'Schedules' in red. On the left, there is a small sailboat icon. The main content is a bulleted list under the heading 'Schedule'. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '31.17', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
 - A schedule for a set of transactions must consist of all instructions of those transactions
 - Must preserve the order in which the instructions appear in each individual transaction
- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
 - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement

So, for doing this we create what is called a schedule? A schedule is a sequence of instructions that specify, the chronological, or the time wise order in which instructions of concurrent transactions are executed. So, what is what will the schedule will have? Scheduler will have for a set of it is defined for the set of transactions. And it must consist of all instructions of those transactions. And in a certain order, and what is the basic requirement that in this schedule in this ordering, the original order of instructions in any of this given transaction, you have an individual transaction must be preserved.

But the instructions from different transaction can be interleaved, intermixed in between to prepare the schedule. So, a transaction that successfully completes its execution will perform what is called a **commit** instruction we will more specifically say what is commit a commit instruction, which means successful completion as the last statement that should be the last statement if the committee is not given by default also transactions which have executed successfully are assumed to have executed commit, or if the transaction fails to successfully complete the execution; that means, we will do abort as a last statement ok.

(Refer Slide Time: 20:12)



Schedule 1

PPD

■ Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B
 ■ An example of a **serial** schedule in which T_1 is followed by T_2 :

T_1	T_2	A	B	A+B	Transaction	Remarks
read (A)		100	200	300	@ Start	
$A := A - 50$		50	200	250	T_1 , write A	
write (A)		50	250	300	T_1 , write B	@ Commit
read (B)		45	250	295	T_2 , write A	
$B := B + 50$		45	255	300	T_2 , write B	@ Commit
write (B)						
commit						
	read (A)					Consistent @ Commit
	$temp := A * 0.1$					Inconsistent @ Transit
	$A := A - temp$					Inconsistent @ Commit
	write (A)					
	read (B)					
	$B := B + temp$					
	write (B)					
	commit					

Database System Concepts - 8th Edition 31.18 ©Silberschatz, Korth and Sudarshan

So, let us take an example so, again we are going back to the same example. So, we have two transactions T1 and T2. T1 transfers 50 dollars from A to B as we have seen. And T2 transfers 10 percent of the balance from A to B. So, one transaction debits 50 dollars one transaction debits 10 percent of the account balance of A to B. So, if they are serially executed as you can see here we are serially executing them as in. So, first you first your whole of T1 is executing, and once this has committed, that is successfully ended then T2 is executing.

So, at the beginning if we assume this is just an assumption. If we assume at the beginning that A had 100 dollar and B had 200 dollar, then the sum was 300 dollar. So, if the A is read 100 dollar is read then it becomes 50 then you write this A. So, when you are here at this point, you can see, this is what you will have because A has changed from 100 to 50 because you have debited B nothing has happened on B so, sum is 250. So, you can see that is why I have shown different colors you can see at 250. This state of the database is temporarily inconsistent because the sum has become different from 300.

Then it reads B it reads 200 adds that 250 it writes B. You come to this point where after this write, when the commit is happening after the writing this B. Then T1 has actually completed, and 50 dollars has got transferred to account B, and the sum is again back to 300 so, consistency is preserved. Then transaction 2 starts so, A is read 50 dollars is read

in temporary you compute 10 percent of that 5 dollar you decrement a by 5 dollar and write it back.

So, when you have written it back, you write back 45 dollar. Again, naturally the sum becomes 5 dollar less, the 5 dollar that you have kept in this temp, and this becomes again transitively inconsistent in the process. Then you do the read B, ad that temporary 5 dollar back to B and then you finally, when you write B here you write back from 200 and 50 you have added 5 dollar to 255, and again the sum becomes 300 you are again back to the consistent state.

So, you can see, through this process that when transactions actually happen in a serial manner, this is how things will move on so, which is quite understandable.

(Refer Slide Time: 23:05)

Schedule 2

PPD

■ A serial schedule in which T_2 is followed by T_1 :

T_1	T_2	A	B	A+B	Transaction	Remarks
	read(A) temp := A * 0.1 A := A - temp write(A) read(B) B := B + temp write(B) commit	100	200	300	@ Start	
		90	200	290	T_2 , write A	
		90	210	300	T_2 , write B	@ Commit
		40	210	250	T_1 , write A	
		40	260	300	T_1 , write B	@ Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

Values of A & B are different from Schedule 1 – yet consistent

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

Database System Concepts - 8th Edition

31.19

©Silberschatz, Korth and Sudarshan

So, let us move on let us. So, this is a different schedule you can see, but this is also a serial schedule here what we have assumed that all instructions of T 2 are done first then all instructions of T 1. I am not going through the going through each step you can see what are the consistent, and the temporarily inconsistently states of the database, but at the end the database is in a consistent state.

And you can note that now the end value of a is 40 dollar, and n value of B is 260 dollar. In the previous schedule, the value was 45 dollar, and 255 dollar these two are different. But both of them are actually correct both of them are consistent, because when things

happen in this distributed manner, we have no control in terms of whether that whether first 50 dollars should be debited and then 10 should be debited transferred. Or whether first 10 should be transferred or 50 dollars where will be transferred after that, either of that is a correct consistent state.

So, the different schedules might give you different results that is not of any concern because both of them are possible valid results. But the question is it must finally, have a consistent state of the database so, both of these are consistent.

(Refer Slide Time: 24:20)



Schedule 3

PPD

■ Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is equivalent to Schedule 1

T_1	T_2	T_1	T_2	A	B	$A+B$	Transaction	Remarks
read (A) $A := A - 50$ write (A)		read (A) $A := A - 50$ write (A)		100	200	300	@ Start	
	read (A) $temp := A * 0.1$ $A := A - temp$	read (B) $B := B + 50$	write (B)	50	200	250	T_1 , write A	
read (B) $B := B + 50$	write (A)	write (B)	commit	45	200	245	T_2 , write A	
commit				45	250	295	T_1 , write B @ Commit	
	read (B) $B := B + temp$	read (A) $temp := A * 0.1$ $A := A - temp$	write (B)	45	255	300	T_2 , write B @ Commit	
	commit	write (A)	commit				Consistent @ Commit	
							Inconsistent @ Transit	
							Inconsistent @ Commit	

Schedule 3 Schedule 1

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

Note – In schedules 1, 2 and 3, the sum “ $A + B$ ” is!

Database System Concepts - 8th Edition 31.29 ©Silberschatz, Korth and Sudarshan

Now, take an interesting example, where schedule 3 where in here if you, if you look at carefully there are few instructions of T_1 which are executed. And then in the temporal order few other instructs, few instructions of T_2 are executed, then again T_1 then again T_2 .

So, the instructions from two different transactions are getting interleaved. And this is what the execution status would be. So, you can see that when you are when T_1 writes A this is where you are 50 dollars has been debited. Then when T_2 writes A subsequently, another 5 dollar is debited so, it becomes 45. So, then you have T_1 again executing and adding B on to that. And by that it is not only that it has gone into an inconsistent, it is it was already in an inconsistent state, but that was transient that was temporary. But now the transaction T_1 has totally completed. It is completed his execution it is at it is commit, but your database is still in an inconsistent state.

So, this is something which is possible, because you are doing an interleaving of the instructions of the two transactions in the schedule. But once you allow the rest of the transaction B this part to complete that is B gets updated and you reach here. And that also has committed. So, your schedule comprised of transaction 1 and transaction 2, when both of them have completed you have again reached state which is consistent. And if you look at the results of what you have achieved you will immediately identify that this doing it doing the transactions according to schedule 3, which is interleaving in this manner is equivalent to this in this manner of interleaving is equivalent to doing them according to in this manner which is schedule 1.

So, you have got a schedule which is equivalent to schedule one. And it is therefore, so, this is just an example to show that it is actually possible to interleave the instructions of 2 transactions, and create a schedule which will still which might in the process have transient or even inconsistent commit states of the database. But finally, when the schedule ends it will it is possible that it will bring you to a consistent state.

(Refer Slide Time: 26:58)

Schedule 4

The following concurrent schedule does not preserve the sum of "A + B"

T ₁	T ₂	A	B	A+B	Transaction	Remarks
read (A)		100	200	300	@ Start	
A := A - 50	read (A)	90	200	290	T2, write A	
	temp := A * 0.1	90	200	290	T1, write A	
	A := A - temp	90	250	340	T1, write B	@ Commit
	write (A)	90	260	350	T2, write B	@Commit
write (B)						
read (B)						
B := B + 50						
write (B)						
commit						
	B := B + temp					
	write (B)					
	commit					

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

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

Database System Concepts - 8th Edition

31.21

©Silberschatz, Korth and Sudarshan

Now, look at again for those transactions look at a different interleaving, a different schedule, again T 1, T2 are involved. But you have now tried to interleave them in a different order. So, earlier the interleaving was done after T1 has done write A, here it is done after he has been the locally debited by 50. And then this part is done and then the

write is happening. And now if you go through the steps I will leave it as an exercise for you in schedule 4.

Now, if you go through the state you will find that when transaction T1 commits ends here, you have an inconsistent state. And finally, even when the schedule ends that T2 has committed. There is A, you are in an inconsistent state somehow that sum of A and B which was 350 has become 300 has become 350 so, 50 dollars as what generated. So, this is so you can see that if you interleave the transactions, then it is quite possible that the transactions will may or may not actually give you a consistent data base.

(Refer Slide Time: 28:07)

Module Summary

- A task is a database is done as a transaction that passes through several states
- Transactions are executed in concurrent fashion for better throughput
- Concurrent execution of transactions raise serializability issues that need to be addressed
- All schedules may not satisfy ACID properties

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

Database System Concepts - 8th Edition 31.22 ©Silberschatz, Korth and Sudarshan

So, here in this module, we have understood the basic tasks that a data base performs database executes; which is in form of a transaction. And we have seen that they must satisfy a set of properties typically called the acid properties, and atomicity, consistency, isolation and durability must be satisfied. And when the transactions are executed in concurrent fashion, we improve the throughput, but the concurrent execution of transaction raise issues of serializability; that is, the concurrent execution that the interleaved schedule of instruction of two or more transactions can give rise to certain effect which violate the acid properties.

And those need to be addressed that certainly inconsistent database is certainly never acceptable. And so that is the basic problem that we have identified which we will have to address in the coming modules.

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

Lecture – 32
Transactions/2: Serializability

Welcome to module 32 of Database Management Systems, from the last module we are discussing about transactions and transaction management.

(Refer Slide Time: 00:27)

The slide has a title 'Module Recap' in red at the top right. On the left, there is a small image of a sailboat on water. A vertical column of text on the left edge reads: 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8th Edition'. The main content area contains a bulleted list: '■ Transaction Concept', '■ Transaction State', and '■ Concurrent Executions'. The bottom right corner features a navigation bar with icons for back, forward, search, and other presentation controls, along with the text '©Silberschatz, Korth and Sudarshan'.

And we have technical look into the basic concept of a transaction the transaction state and the issues.

(Refer Slide Time: 00:35)

The slide is titled "Module Objectives" in red bold font at the top right. At the top left is a small sailboat icon. The title "Module Objectives" is centered above a bulleted list of three items. The footer contains course and copyright information.

Module Objectives

- To understand the issues that arise when two or more transactions work concurrently
- To introduce the notions of Serializability that ensure schedules for transactions that may run in concurrent fashion but still guarantee and serial behavior
- To analyze the conditions, called conflicts, that need to be honored to attain Serializable schedules

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur-2018

Database System Concepts - 8th Edition

PPD

32.3

©Silberschatz, Korth and Sudarshan

In concurrent execution and in this module, we will look try to understand, what are the very specific issues that happen when two or more transactions work concurrently we have seen that now it is possible that they execute in a schedule, which would not let us preserve the acid properties.

So, we want to introduce the very basic concept of making sure that such concurrent execution schedules are acceptable, and those are the notions of serializability. And we will analyze different conditions called conflicts that need to be honored to attend the serializability of the schedules.

(Refer Slide Time: 01:17)

Module Outline

PPD

- Serializability
- Conflict Serializability

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

32.4

©Silberschatz, Korth and Sudarshan

So, serializability is the main topic to discuss.

(Refer Slide Time: 01:21)

Serializability

PPD

- **Basic Assumption** – Each transaction preserves database consistency
- Thus, serial execution of a set of transactions preserves database consistency
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
 1. **conflict serializability**
 2. **view serializability**

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

32.6

©Silberschatz, Korth and Sudarshan

So, to understand serializability we make a basic assumption, we make an assumption that every transaction by itself preserves the database consistency. That is, it starts in a consistent state of the database. And through the execution of its instructions in the order given it leaves the database in a consistent state, that is satisfied in each and every transaction. So, we can conclude that, if we serially execute a set of instruction set of transactions, then the consistency of the database will always be preserved.

Now, the problem happens, and as we have seen in the last module, that problems happen when possibly concurrent transactions happen. And we may execute may be executing the instruction in an order which leads to the violation of acid properties, the consistency in particular. So, we say that a concurrent schedule is serializable, if there is a there is some serial schedule, you say what is the serial schedule serial schedule is where the transactions are executed one after the other.

So, if you have 2 transactions in the concurrent system, then if I do T 1 then I do T 2 it is a serial schedule. If I do T2 and then I do T1 it is a serial schedule as well. So, if I have a concurrent schedule, if you refer back to the last module schedule 3; where the instructions of T1 and T2 are interleaved, then it is it will have to be equivalent to a serial schedule either T1 after T2 or T2 after T1. Different forms of schedule equivalence is used one is called conflict serializability, and the other is called view serializability. In the present module we will first discuss conflict serializability.

(Refer Slide Time: 03:19)

The slide has a header 'Simplified view of transactions' with a sailboat icon. The main content is a bulleted list of assumptions:

- We ignore operations other than **read** and **write** instructions
 - Other operations happen in memory (are temporary in nature) and (mostly) do not affect the state of the database
 - This is a simplifying assumption for analysis
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes
- Our simplified schedules consist of only **read** and **write** instructions

At the bottom left is a video thumbnail of a speaker, and at the bottom right is a navigation bar.

Now, we make now a transaction may have all varied kinds of instructions, but we make an assumption that we will ignore anything other than any instruction other than the read and write instruction. Because other operations like we saw an operation where an account is debited by 50 or account is credited. So, you subtract 50 you add 50 you multiply by 0.1 or things like that are all operations that happen in the local buffer in the memory, and never temporary in nature and mostly they do not affect the state of the

database, because you have read the data do the changes write it back. So, it is a read and write that actually are important for that maintaining the consistency after database. So, that simplifies our process of analysis to a good extent.

So, this is so, we assume that between every read and write or read and read write and write and so on, the database the transactions may be doing arbitrary computations, which are all in the local buffer and do not affect the state. So, we can make this assumption that our shift schedules consist only of read and writing.

(Refer Slide Time: 04:31)

The slide has a title 'Conflicting Instructions' in red at the top right. On the left is a small image of a sailboat on water. The main content area contains the following text:

Let I_i and I_j be two instructions of transactions T_i and T_j respectively. Instructions I_i and I_j conflict if and only if there exists some item Q accessed by both I_i and I_j , and at least one of these instructions wrote Q .

- 1. $I_i = \text{read}(Q)$, $I_j = \text{read}(Q)$. I_i and I_j don't conflict
- 2. $I_i = \text{read}(Q)$, $I_j = \text{write}(Q)$. They conflict
- 3. $I_i = \text{write}(Q)$, $I_j = \text{read}(Q)$. They conflict
- 4. $I_i = \text{write}(Q)$, $I_j = \text{write}(Q)$. They conflict

Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them

- If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule

At the bottom left is a video thumbnail of a professor. At the bottom center is the text 'Database System Concepts - 8th Edition'. At the bottom right is the text 'Silberschatz, Korth and Sudarshan'.

Now, we say that suppose I_i and I_j , 2 instructions for belonging to transaction T_i and transaction T_j . So, there are two transactions T_i and T_j , T_i has an instruction I_i , T_j has an instruction I_j and we say that I_i ad I_j this instruction will conflict, if and only if there is some item Q , that is some data entity Q ; which both I_i and I_j are trying to access. And at least one of these instructions try to write.

So, these two instructions from true transactions are trying to manipulate the same data item, and at least one of them is trying to write. If that happens then we say that I_i and I_j these two instructions are conflicting. So, you can naturally enumerate the four possibilities, if both of them are reading their own conflict. If it is read write, write read, write All of them are cases of conflict.

So, naturally intuitively, you can figure out that since the write changes are value that if there is a conflict between these two instructions then there must be a fixed temporal order between them. So, if I_i and I_j are consecutive in a schedule and they do not conflict. Then we can interchange the temporal order of I_i and I_j , that will also not make a difference, because they do not conflict. But if they conflict I cannot make the change in their ordering.

(Refer Slide Time: 06:18)

The slide has a title 'Conflict Serializability' in red at the top right. To its left is a small sailboat icon. On the left side, there is vertical text: 'SWAYAM-NPTEL-NOJC-MOOCs Instructor: Prof. P. Deshpande, IIT Kanpur - Jan-Apr- 2018'. At the bottom left is a video player showing a man speaking. The bottom right corner shows the copyright notice '©Silberschatz, Korth and Sudarshan'.

Conflict Serializability

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**
- We say that a schedule S is **conflict serializable** if it is conflict equivalent to a serial schedule

So, that gives rise to the notion of conflict serializability. So, we say if a schedule S can be transformed into another schedule S' by a series of swaps of non-conflicting instructions, then S and S' that conflict equivalent. So, what are you saying? That we have two one schedule S , and we will swap non-conflicting instruction, possibly since non-conflicting instructions that occur side by side. And if by doing this, if I can create the schedule S primed, then I will say S and S' that conflict equivalent. But if S and S' that, I cannot transform S into S' by just swapping non-conflicting instructions, then they are not conflict equivalent.

The second definition to keep in mind is a schedule S is conflict serializable, if it is conflict equivalent to a serial schedule, what is the serial schedule? Just to remind you serial schedule is one where the transactions are happened one after the other in a serial manner. So, all instructions of one transaction complete, then all instructions of the second transaction complete, then all instructions of the third transaction complete and so

on. So, if a schedule is conflict serializable; that is, if in a schedule. I can swap non-conflicting instructions. And make it into a serial schedule, and then I will say that the given schedule is a conflict serializable schedule ok.

(Refer Slide Time: 08:00)

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6 – a serial schedule where T_2 follows T_1 , by a series of swaps of non-conflicting instructions.
 - Swap $T_1.\text{read}(B)$ and $T_2.\text{write}(A)$
 - Swap $T_1.\text{read}(B)$ and $T_2.\text{read}(A)$
 - Swap $T_1.\text{write}(B)$ and $T_2.\text{write}(A)$
 - Swap $T_1.\text{write}(B)$ and $T_2.\text{read}(A)$
- These swaps do not conflict as they work with different items (A or B) in different transactions.
- Therefore, Schedule 3 is conflict serializable:

T_1	T_2	T_1	T_2	T_1	T_2
read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)

Schedule 3 Schedule 5 Schedule 6

So now let us it is time for a number of examples to understand this better. So, we had seen schedule 3, will have to refer to the earlier module 4 schedule 3. Sir, no not I am sorry this is just abstracted form of that; not the actual one because in the in the earlier schedule 3 we had shown all the complete other computations also, but the read writes are the same.

Now, that this schedule 3 can be converted to so, this is where you have schedule 3, and you can easily see that the part of transaction T_1 then a part of transaction T_2 . So, schedule 3 is not a serial schedule, but if you can swap non conflicting instructions, then you are able to convert this into this schedule which if we are calling a schedule 6. Where all instructions of T_1 is followed by all instructions of T_2 which is a serial schedule.

So, since this can be done, we will say it is conflict serializable schedule 3 is conflict serializable and just to see how that happens. So, you start here let me erase this marks and start here. So, here if I look into these two instructions, which are the consecutive instructions in schedule 3 I can swap them; that is, I can do read B first and then do read A, I can swap read B and write A read B, and write A can be swapped.

Once I have done that, then I can swap read B with read A. It has become before right A can swap it with, because read B and write A, or write B read B and read A these do not conflict their non-conflicting instruction. Why read B and write A and non-conflicting, because they are not reading and writing to the same data item. Why read B and read A are non-conflicting, they are accessing the same data item, but both of them are read there is no write. So, I can swap so, this is the second one I can. So, once I do that read B will come here and write (A), read (A), write(A)will come down.

Then again, I can see that write B can be swapped with write A. Both are rights, but referring to different data items. Similarly, write B then can be swapped with read A, because they are again referring to different data items. So, I can do this and then these will also come up. So, I will eventually after these 4 swaps, this whole schedule 3 will transform into this serial 6, and we get a serial schedule.

So, we will say that schedule 3 is conflict serializable. That is the basic concept that we are trying to establish here.

(Refer Slide Time: 11:02)

T_3	T_4
read (Q)	
	write (Q)
write (Q)	
	read (Q)

■ We are unable to swap instructions in the above schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$, or the serial schedule $\langle T_4, T_3 \rangle$

Just as very simple example suppose you had two transactions T_3 and T_4 , and you have this situation. Now is it conflict serializable it is not. Because to make it conflict serializable. I need to either swap write(Q) of T_3 with write(Q) of T_4 which is not possible because these are conflicting instructions, they both access the same data item Q and they both are write.

The other option is I could swap read(Q) in T₃ and write(Q) in T₄, that they are also conflicting because they access the same data item and one of them is write. So, I cannot do either of this swaps which mean, that I cannot find a conflict equivalent schedule for this schedule; either to T₃ T₄ or to T₄ T₃. It is not this schedule is not conflict equivalent to either one of them.

So, this schedule is not conflict serializable, this is the core concept. So, if you go through different examples and try to understand this at the very beginning, then in terms of the transaction management the whole study of transaction management you will have very easy progress.

(Refer Slide Time: 12:37)

Example: Bad Schedule

Consider two transactions:

Transaction 1	Transaction 2
UPDATE accounts	UPDATE accounts
SET balance = balance - 100	SET balance = balance * 1.005
WHERE acct_id = 31414	

In terms of read / write we can write these as:

- Transaction 1: $r_1(A), w_1(A) // A$ is the balance for acct_id = 31414
- Transaction 2: $r_2(A), w_2(A), r_2(B), w_2(B) // B$ is balance of other accounts

Consider schedule S:

- Schedule S: $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
- Suppose: A starts with \$200, and account B starts with \$100
- Schedule S is very bad! (At least, it's bad if you're the bank!) We \$100 from account A, but somehow the database has that our account now holds \$201!

Source: <http://www.cburch.com/cs/340/reading/serial/>

SWAYAM, NPTEL, NOC, INOC Instructor: Prof. P. P. Desai, IIT Kanpur - June-Apr. 2018

Database System Concepts - 8th Edition

32.13

©Silberschatz, Korth and Sudarshan

So, let us let me show you number of other bad schedules, and let me a little bit more complex examples.

So, consider two transactions transaction 1 here. Update an account, where the account id is 31414 a specific account and balance is debited by 100. So, it is debiting 100 from the balance. Where in the transaction 2, you update accounts where balance is changed to balance times 1.005 which means that we are giving a 0.5 percent interest, and here there is no where clause. So, transaction 2 actually changes does this balance change in all the accounts, whereas, transaction 1 makes this debit in only one account.

Let see what will happen in terms of them. So, let us first try to write out transaction 1 and transaction 2, the first in the read write Abstracted form. So, transaction 1 it is working only on one account let us call it account A. So, what does it do? It has to set the balance to debit 100. So, it has to read so, this is r_1 by $w_1(A)$, we mean that it is read the subscript here refers to the transaction number.

So, r_1 stands for r stands for read, 1 stands for transaction 1. So, it is read by transaction 1. And what are we reading? We are reading the account balance A, let us arbitrarily we are calling it A. And then what we will have to do? After having debited that locally we will have to write it back so that the change has happened. So, $r_1(A)$ followed by $w_1(A)$ is transaction 1 which is being shown on the left. So, I have shown you from the actual sql statement, how can you make an abstraction of the read write that we use in terms of reasoning about the serializability.

In contrast, if you look at transaction 2, naturally transaction 2 does not have a where clause. So, it performs this balance update on each of the accounts. So, it will also perform this balance update on the account A or account balance A rather, that we assumed in the transaction 1. So, we model that by saying that naturally for changing the balance from balance times 1.005, we need to read A if the read is done in transaction 2. So, that is $r_2(A)$ and write it back. So, that is $w_2(A)$ and then I assume that B is some other account. There may be one more account there may be 100 thousand more account, but so far as serializability are concerned, these are all other different accounts from A. So, we symbolically just consider one; that is, some other account B other than the balance a and naturally to do the change here or do the update here will have to read B r to be and w to B. So, these are the two transactions in the simplified form that we have to analyze.

Now, let us consider A so, we have between transaction 1 and transaction 2 we have 6 instructions. So, we produce a schedule 6, where these 6 instructions are interleaved. And we satisfy the basic constraint that the instructions of every transaction occur in the same order in which they existed. So, r_1 precedes w_1 , r_1 precedes w_1 in this schedule. R 2 A precedes $w_2(A)$, $w_2(A)$ precedes $r_2(B)$, $r_2(B)$ precedes $w_2(B)$ and so on. So, their original ordering is maintained, but we have an interleaved schedule called S. And then on the on the write if you look at here on the write this is schedule S.

So, in the write we are saying, that let us say that A starts with 200 dollar, and B at the beginning is 100 dollar. So, what will happen you will read? You will read here this is the first thing $r_1(A)$. So, 200 is read then $r_2(A)$. So, what happens if $r_2(A)$ is read Again 200 is read. And then you do $w_1(A)$. So, what is $w_1(A)$? $w_1(A)$ is the write of the transaction 1. So, transaction 1 writes after debiting this balance this is the intermediate computation.

So, when transaction 1 writes, it writes based on the value that it had read in $r_1(A)$; which was 200 then 100 debited. So, it writes back 100. Then the next is $w_2(A)$. So, what will $w_2(A)$ do? $w_2(A)$ will write back the write back $w_2(A)$ is here, will write back the result of the computation in transaction 2 based on what it read in the $r_2(A)$. $r_2(A)$ had written had read 200, we hear and therefore, if you multiply 200 by this factor it becomes 201. So, $w_2(A)$ will write 201.

So, naturally E as $w_2(A)$ has changed the value of A after $w_1(A)$ naturally the final value of A will be 201. Then you have r to be w to be which reads 100 makes this balance change by 1.005, it becomes 100.5. So, this is what we will have when actually this schedule completes. So, if I mean just this look at what has happened, it has I have debited 100 dollar from account A which was transaction 1, but and here I had started with 200 dollar. But at the end what I have according to the schedule is account A has a balance which is 201 dollar. Whereas, it should have had a balance which should have been 100 dollar, the balance in account B is fine. But it shows that 101 dollar more in account A.

So, naturally the bank is going to get bankrupt very soon if such scheduled are allowed. So, this schedule is an incorrect inconsistent schedule, it is a bad schedule, let us take other examples.

(Refer Slide Time: 19:18)

The slide features a title 'Example: Bad Schedule' with a sailboat icon. It contains text about ideal schedules, serial schedules, and a bad schedule (T). It includes a table comparing two schedules (S and T) with their transaction steps and final values for A and B.

Example: Bad Schedule

■ Ideal schedule is serial: (A = \$200, B = \$100)

Serial schedule 1: $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B) // A = 100.50, B = 100.50$

Serial schedule 2: $r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A) // A = 101.00, B = 100.50$

■ We call a schedule **serializable** if it has the same effect as some serial schedule regardless of the specific information in the database.

■ As an example, consider Schedule T, which has swapped the third and fourth operations from S:

- Schedule S: $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
- Schedule T: $r_1(A), r_2(A), w_2(A), w_1(A), r_2(B), w_2(B)$

■ By first example, the outcome is the same as Serial schedule 1. But that's just a peculiarity of the data, as revealed by the second example, where the final value of A can't be the consequence of either of the possible serial schedules.

S nor T are serializable.

	A	B	A	B
initial:	100.00	100.00	(initial):	200.00
$r_1(A):$			$r_1(A):$	
$r_2(A):$			$r_2(A):$	
$w_2(A):$	100.50		$w_2(A):$	201.00
$w_1(A):$	0.00		$w_1(A):$	100.00
$r_2(B):$			$r_2(B):$	
$w_2(B):$	100.50		$w_2(B):$	100.50

Schedule T

Source: <http://www.cburch.com/cs/340/reading/serial/>

SWAYAM/NPTEL NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

32.14

©Silberschatz, Korth and Sudarshan

Now let us ask what is the ideal schedule, what is ideally what should happen. Ideally naturally we can have we will have serial schedules, there are two transactions. So, there are two possible serial schedules that can happen that is first T_1 happens, then whole of T_2 happens. I am sorry, first T_1 and then whole of T_2 . Or first whole of T_2 and then T_1 . And if you go through the steps, assuming that A initially is 200 dollar and B is 100 dollar, these are the possible results that you see naturally. As I had mentioned earlier also, the different ordering different schedule might give you different results, but both of them are correct, because any one of them will happen, but both are consistent. Either debit has first happened, then the interest credit or interest credit it has first happened and then the debit.

So, either of these schedules are acceptable, but what we got as a schedule S in the last case are not acceptable. So, we will call it you will serializable, if it has the same effect, as some of the one of the two schedules that we have here. Then we will say that is it this is serializable schedule. So, again we create another example schedule T here. So, what we do? We take the schedule S which we saw was bad, and we interchange, these two we do $w_2(A)$ first and $w_1(A)$ next.

Now, you see very interesting things will happen. So now, you focus on this part, on the left part of schedule T where we are assuming that A and B both have 100 dollar to start with. And then go through these steps r_1 is in transaction 1 r_2 is in transaction 2, then w_2

happens so, the interest is credit 100.5. And then what has happened? w_2 after that $w_1(A)$ so, whatever was written here is debited and written back. So, whatever was read there is 100 dollar. So, you debit 100 dollar it becomes 0.

So, A has become 0, and then you have the B which goes on correctly. So, things look like that it appears that we are perfectly ok. So, by the first example the outcome is same as a serial schedule one. And so, we might just think that things have been good, but this is just incidental based on the particular values. Now let us consider another execution by the same schedule which makes use of this value 200 and 100.

Now, as it with 200 and 100 and we do w_2 followed by w_1 . So, when $r_2(A)$ is followed by w_2 , $r_2(A)$ 100 read 200 and that 10, 1.005 or that kind of interest is given then it becomes 201. And then r_1 then you have w_1 . Now what does $w_1(A)$ changes? r_1 had read 200, and from that you have subtracted 100. So now, you have as $w_1(A)$, you have 100 input. And from this you have B certainly does not change.

So, if you look into that, now you can see that he has A value which is 100; which certainly if you look back. So, 200 and 100 are the values that we had assumed here, and you can see that in neither of the schedule A can have a value, which is 100 dollar as we have found here. It can either be 100.50 or it can be 101, but you have got a value 100. So, even though with some data, a schedule might look like serializable, but it actually is not and it needs to be properly established that schedule is serializable.

(Refer Slide Time: 24:00)

Example: Good Schedule

■ What's a non-serial example of a serializable schedule?

- We could credit interest to A first, then withdraw the money, then credit interest to B:
- Schedule U: $r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B) // A = 101, B = 100.50$

■ Schedule U is conflict serializable to Schedule 2:

Schedule U: $r_2(A), w_2(A), r_1(A), \underline{w_1(A)}, r_2(B), w_2(B)$
 swap $w_1(A)$ and $r_2(B)$: $r_2(A), w_2(A), r_1(A), \underline{r_2(B)}, \underline{w_1(A)}, w_2(B)$
 swap $w_1(A)$ and $w_2(B)$: $r_2(A), w_2(A), r_1(A), r_2(B), w_2(B), w_1(A)$
 swap $r_1(A)$ and $r_2(B)$: $r_2(A), w_2(A), r_2(B), \underline{r_1(A)}, \underline{w_2(B)}, w_1(A)$
 swap $r_1(A)$ and $w_2(B)$: $r_2(A), w_2(A), r_2(B), \underline{w_2(B)}, \underline{r_1(A)}, w_1(A)$: Schedule 2

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

Source: <http://www.cburch.com/cs/340/reading/serial/>

So, neither S naught T are serializable, yet another schedule U this again. So, you can see that transaction 1 is happening instruction of transaction 1 is happening somewhere in the middle. With transaction 2 and this is what you get. So, if you if you look back as to earlier case. You will find that this is same as scheduled 2. So, this is same as scheduled 2.

So, again my the data it looks like that this is correct, but we have to actually establish that this is correct. So, we can establish say that by proving that schedule 2 is, I am sorry, schedule 2 is conflict serialize, schedule U is conflict serializable to schedule 2. How we do that? We keep on swapping the non-conflicting instructions. This is one we start with, and we swap w_1 with $r_2(B)$ this is possible they are referring to two different data items. Then we swap w_1 with w_2 again different data items. Then we swap r_1 with r_2 again different data items and also, they are both of them are read. And finally, we swap $r_1(A)$ with $w_2(A)$, $r_1(A)$ with $w_2(B)$ and we get this, and now you can see that this is transaction 2 followed by transaction 1 which is scheduled 2. Which is indeed a serial schedule and we have been able to transform schedule U into a conflict equivalent schedule 2 which is serial.

So, we will say that while our earlier attempts on schedule S and schedule T were not serializable schedule U is serializable.

(Refer Slide Time: 26:09)

The slide has a title 'Serializability' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points related to serializability:

- Are all serializable schedules conflict-serializable? No.
- Consider the following schedule for a set of three transactions.
 - $w_1(A), w_2(A), w_2(B), w_1(B), w_3(B)$
- We can perform no swaps to this:
 - The first two operations are both on A and at least one is a write;
 - The second and third operations are by the same transaction;
 - The third and fourth are both on B at least one is a write; and
 - So are the fourth and fifth.
 - So this schedule is not conflict-equivalent to anything – and certainly not any serial schedules.
- However, since nobody ever reads the values written by the $w_1(A)$, $w_2(B)$, and $w_1(B)$ operations, the schedule has the same outcome as the serial schedule:
 - $w_1(A), w_1(B), w_2(A), w_2(B), w_3(B)$

At the bottom, there is a footer with the text 'Source: <http://www.cburch.com/cs/340/reading/serial/>' and a navigation bar with icons for back, forward, search, etc.

So, naturally all serializable schedules are they conflict serializable. No, for example, here I have given. So, here what we are trying to highlight is a schedule may be serializable, but it may not be conflict serializable. So, conflict serializability is a stronger notion. So, here I have given a small example which I leave to you to go through in detail and understand where it is not possible to show that it is conflict serializable in the sense you cannot there are three transactions here w_1 w_2 and w_3 . And you cannot swap non-conflicting instructions in this schedule and convert it into a serial schedule.

So, serial schedule here will mean, $T_1, T_2, T_3, T_1, T_3, T_2, T_2, T_1, T_3$ like that. Any of the 6 possibilities, you cannot convert this in a conflict equivalent manner to any of those 6 serial schedules. But this actually is a serial schedule, because very interestingly even though there are multiple writes, but in between there are no reads. So, you can you can easily reason, that the values have actually not changed ok. So, this is on the basic notion of serializability.

(Refer Slide Time: 27:36)



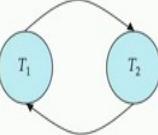
SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018



Database System Concepts - 6th Edition

Precedence Graph

- Consider some schedule of a set of transactions T_1, T_2, \dots, T_n
- **Precedence graph**
 - A direct graph where the vertices are the transactions (names)
 - We draw an arc from T_i to T_j if the two transactions conflict, and T_i accessed the data item on which the conflict arose earlier
 - We may label the arc by the item that was accessed
 - Example





©Silberschatz, Korth and Sudarshan

Now, the question naturally is how do I detect, if a schedule is serializable. So, the process is to construct a what is called a precedence graph. So, if I have a set of transactions, then I construct a graph is a directed graph where the vertices are the transactions their names. And we will draw an graph from T_i to T_j , that is my graph means there will be an edge is a directed edge. If these 2 transactions T_i and T_j are

730

conflicting. So, if T_i T_j conflict there will be edge between them. And the edge will be from T_i to T_j , if T_i access the data item which conflict with T_j . So, if T_i is a head is earlier, then we will draw the arc from T_i to T_j , otherwise it will be from T_j to T_i . And we may also annotate label the arc by the item on which item that is being accessed.

(Refer Slide Time: 28:47)

Testing for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph
 - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
 - That is, a linear order consistent with the partial order of the graph.
 - For example, a serializability order for the schedule (a) would be one of either (b) or (c)

(a)

(b)

(c)

SWAYAM-NETEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr - 2018
Database System Concepts - 8th Edition
32.18 ©Silberschatz, Korth and Sudarshan

So, this could be A so, possible what is called the precedence graph. So, it is a schedule is conflict serializable, if and only if it is precedence graph is acyclic. Naturally, if there is a cycle then; that means, that any of the like we have here, if there if it is a cyclic like this then it is possible that I can actually do a topological ordering of these nodes, and we can find a serial schedule. But if it has a cycle then naturally, I i cannot put any of the transactions on the cycle at the beginning and put the others on the later part things will; obviously, always conflict.

So, we can easily these are details of the algorithms, cycle detection can be done very easily. Either in n square time in a simple manner or when n plus E time where E is a number of edges. So, the precedence if the precedence graph is acyclic the serializability order will be obtained by simple topological sorting. I am not discussing what these algorithms are I would expect that you know if you do not please look up in algorithms book.

(Refer Slide Time: 30:07)

The slide has a header 'Testing for Conflict Serializability' with a sailboat icon. On the left, vertical text reads 'SWAYAM-NIETL-NOOC Instructor: Prof. P P Desai, IIT Kanpur', 'Date: Jan-Apr, 2018', and 'PPD'. The main content is a bulleted list:

- Build a directed graph, with a vertex for each transaction.
- Go through each operation of the schedule.
 - If the operation is of the form $w_i(X)$, find each subsequent operation in the schedule also operating on the same data element X by a different transaction: that is, anything of the form $r_j(X)$ or $w_j(X)$. For each such subsequent operation, add a directed edge in the graph from T_i to T_j .
 - If the operation is of the form $r_i(X)$, find each subsequent write to the same data element X by a different transaction: that is, anything of the form $w_j(X)$. For each such subsequent write, add a directed edge in the graph from T_i to T_j .
- The schedule is conflict-serializable if and only if the resulting directed graph is acyclic.
- Moreover, we can perform a topological sort on the graph to discover the serial schedule to which the schedule is conflict-equivalent.

At the bottom, there is a video player showing a man speaking, the text 'Database System Concepts - 8th Edition', the page number '32.19', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, to test for conflict serializability; the steps will be build the directed graph. Then go through each operation of shall, you look at each operation read or write. If the operation is a write, then find so, if it is $w_i(X)$, then find what is happening with data this data element X in different transactions that exists later on that instructions exist later on.

If there is some $r_j(X)$ or some $w_j(X)$, either in this was in transaction I_j , in transaction some transaction j if there is a read X or if there is a write of X , then there will be a directed graph age from T_i to T_j . This is what I said earlier. On the other case if your operation is of the from r_i J if it is a read operation then all that you need to look for is only a right on this X on the different transaction. And then you will have a naturally if you if your current operation is read and you do not find a write there may be other reads on X then you do not add any conflict edge.

So, on this graph the schedule is conflict serializable if it is acyclic, and we will do topological sort to get that as I have.

(Refer Slide Time: 31:28)

Testing for Conflict Serializability

- Consider the following schedule:
 - $w_1(A), r_2(A), w_1(B), w_3(C), r_2(C), r_4(B), w_2(D), w_4(E), r_3(D), w_5(E)$
- We start with an empty graph with five vertices labeled T_1, T_2, T_3, T_4, T_5 .
- We go through each operation in the schedule:
 - $w_1(A)$: A is subsequently read by T_2 , so add edge $T_1 \rightarrow T_2$
 - $r_2(A)$: no subsequent writes to A , so no new edges
 - $w_1(B)$: B is subsequently read by T_4 , so add edge $T_1 \rightarrow T_4$
 - $w_3(C)$: C is subsequently read by T_2 , so add edge $T_3 \rightarrow T_2$
 - $r_2(C)$: no subsequent writes to C , so no new edges
 - $r_4(B)$: no subsequent writes to B , so no new edges
 - $w_2(D)$: C is subsequently read by T_2 , so add edge $T_3 \rightarrow T_2$
 - $w_4(E)$: E is subsequently written by T_5 , so add edge $T_4 \rightarrow T_5$
 - $r_3(D)$: no subsequent writes to D , so no new edges
 - $w_5(E)$: no subsequent operations on E , so no new edges
- We end up with precedence graph
- This graph has no cycles, so the original schedule must be serializable. Moreover, since one way to topologically sort the graph is $T_3-T_1-T_4-T_2-T_5$, one serial schedule that is conflict-equivalent is
 - $w_3(C), w_1(A), w_1(B), r_4(B), w_4(E), r_2(A), r_2(C), w_2(D), r_3(D), w_5(E)$

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

Source: <http://www.cs.cmu.edu/~cga/210/lesson10serial/>

Database System Concepts - 8th Edition 32.20 ©Silberschatz, Korth and Sudarshan

So, here what I have done is I have actually taken a little bigger example, where you can see that at the beginning here. I have given a schedule which has 5 transactions. And it has A B C D E, 5 different data elements, and variety of read write happening on them. So, based on that, you start with an empty graph having so, your graph will have 5 nodes because these are the transactions. And then you go through the schedule, you start with the very first one $w_1(A)$. So, a is the data item you are looking at and then you see who is doing it. So, you see that well a is read by T 2.

So, there is a conflict. So, you will add an edge $T_1 T_2$ so, this edge gets added. Then is to have $r_2(A)$, and you find look for A, A, A, A, A there is no A so, there is no subsequent write. So, there is no new edge then you have $w_1(B)$, $w_1(B)$ and if you look for you have $r_4(B)$; so r_4 that this transaction 4 is reading it later on. So, $w_1(A)$ B subsequently read by T_4 so, there is a conflict. So, you add the edge T_1, T_4, T_1, T_4 . You proceed in this way, you can work it out in full. And when you come to the end you have constructed this particular graph which is the precedence graph. And you can very easily see that this precedence graph is acyclic there is no cycle here. And therefore, the original schedule is serializable. And what is that what is the order in which you find out what is the corresponding serial schedule for that you do a topological sort.

So, by topological sort which will mean this have no predecessor. So, any one of them can be the first node other one can be the next node. So, it could be $T_3 T_1$ or $T_1 T_3$. So, let

us say T_3 T_1 then T_3 T_1 has happened. So, I can put any one of T_2 or T_4 after that. Here I put T_4 then T_2 . So, up to this and then finally, T_5 . So, this is one possible serial schedule to which this given schedule is conflict serializable. And so, the actual serial schedule. So, if you do this schedule, you will get a result which is a result of this serial schedule which is T_3, T_1, T_4, T_2, T_5 . It is also actually this channel is conflict serializable to several other schedule because you can do this topological sorting in various different manners, you could have started with T_1 and then do T_3 and then do the rest. You could have done T_3, T_1 , and then instead of doing T_4, T_2 , you could do T_2, T_4 .

So, you will get a number of, but having one equivalent serial schedule. One conflict equivalent serial schedule is enough to prove the serializability of a schedule. Now so, based on that you say that this particular schedule is conflict serializable, and it will be safe to execute the interleaved instructions of the 5 different transactions in this manner in the schedule, and we will always have a consistent result.

(Refer Slide Time: 35:18)

Module Summary

- Understood the issues that arise when two or more transactions work concurrently
- Learnt the forms of serializability in terms of conflict and view serializability
- Acyclic precedence graph can ensure conflict serializability

SWAYAM: NIITL-NIITL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr- 2018

Database System Concepts - 8th Edition 32.21 ©Silberschatz, Korth and Sudarshan

So, here in this module, you have understood the issues that arise in terms of concurrency when two or more transactions work concurrently. And very specifically we have learnt about different forms of serializability. In this module we have talked of conflict serializability, view serializability we will take up later on. And we have seen an algorithm, simple algorithm, based on the acyclic precedence graph, which will allow you to ensure that a given schedule is conflict serializable or not.

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

Lecture – 33
Transactions/3 : Recoverability

Welcome to module 33 of Database Management Systems. This is on transactions again there is a third and closing module on transactions and, we will discuss recoverability issues and some more of the serializability issues in this module.

(Refer Slide Time: 00:40)

The slide is titled "Module Recap" in red at the top right. On the left, there is a small image of a sailboat on water. The footer contains the text "SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018", "Database System Concepts - 8th Edition", "33.2", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the bottom right.

In the last module we have talked at length about serializability and specifically, we looked at what is known as conflict serializability and the algorithm to detect that. ah

(Refer Slide Time: 00:50)

Module Objectives

- What happens if system fails while a transaction is in execution? Can a consistent state be reached for the database? Recoverability attempts to answer issues in state and transaction recovery in the face of system failures
- Conflict serializability is a crisp concept for concurrent execution that guarantees ACID properties and has a simple detection algorithm. Yet only few schedules are Conflict serializable in practice. There is a need to explore – View Serializability – a weaker system for better concurrency

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: 2018

Database System Concepts - 8th Edition

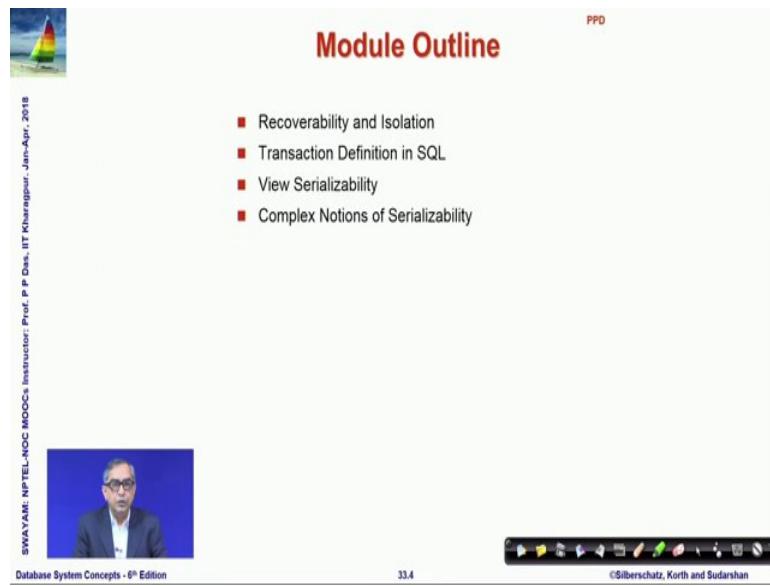
33.3

©Silberschatz, Korth and Sudarshan

Now, we would bring in another perspective is if while a transaction is in execution what if the system would fail, the failure may be due to hardware software, various different reasons power outage, disk crash and so on. So, why when that happens the database is likely to come into an inconsistent state. So, we would like to discuss how to recover from that inconsistent state and bring it back to a consistent state.

We would also look at that going forward from conflict serializability, what are the other notions of serializability, that can be used to serialize transactions and we will look at a weaker definition of serializability known as view serializability, which can serialize more schedules than what conflict serializability can give us.

(Refer Slide Time: 01:51)



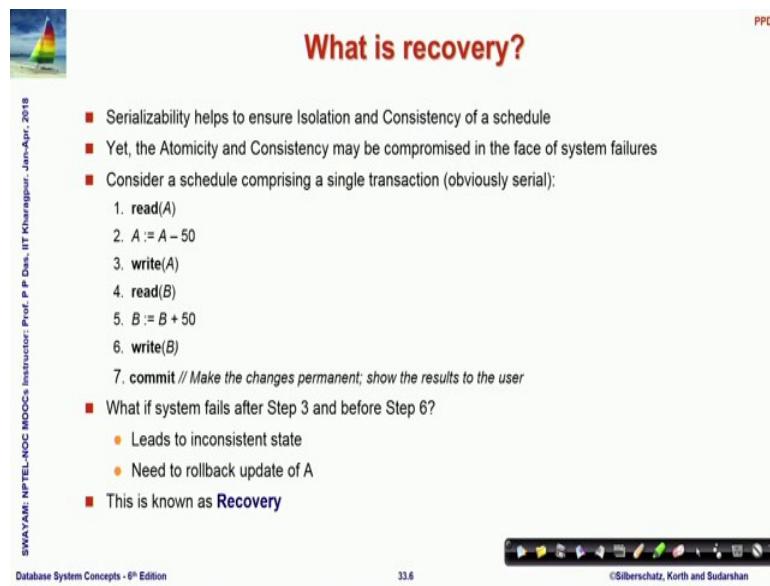
The slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is a vertical column of text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande", "IIT Kharagpur", and "Jan-Apr., 2018". In the center, there is a video frame showing a man with glasses and a blue background. The right side contains a bulleted list of topics:

- Recoverability and Isolation
- Transaction Definition in SQL
- View Serializability
- Complex Notions of Serializability

At the bottom, it says "Database System Concepts - 8th Edition", "33.4", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

So, these are the topics to discuss and we start with recoverability and isolation.

(Refer Slide Time: 01:57)



The slide is titled "What is recovery?" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is a vertical column of text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande", "IIT Kharagpur", and "Jan-Apr., 2018". The central part of the slide contains a bulleted list of points about recovery:

- Serializability helps to ensure Isolation and Consistency of a schedule
- Yet, the Atomicity and Consistency may be compromised in the face of system failures
- Consider a schedule comprising a single transaction (obviously serial):
 1. `read(A)`
 2. `A := A - 50`
 3. `write(A)`
 4. `read(B)`
 5. `B := B + 50`
 6. `write(B)`
 7. `commit // Make the changes permanent; show the results to the user`
- What if system fails after Step 3 and before Step 6?
 - Leads to inconsistent state
 - Need to rollback update of A
- This is known as **Recovery**

At the bottom, it says "Database System Concepts - 8th Edition", "33.6", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

So, what we have done is we have seen the serializability help us, if we think in terms of the acid properties that we started by defining as the desirable properties of the transactions, we have seen that the serializability significantly helps us to achieve isolation and consistency of a schedule, yet the atomicity and consistency may be compromised, if there is a system failure.

So, we had talked about this example a bit earlier again let us take a look. So, this is a transaction where an amount of 50 dollar is being transferred from account A to account B. So, he first read debit and then write on account A and then read credit and write to account B and we have added a 7th instruction, which is commit and I will talk more about that in this module which makes that changes to A and B permanent and shows a result to the user as well.

Now, what happens if the system fails between step 3 and after step 3 when A has been written and before step 4 step 6 when B has finally, been written. So, naturally 50 dollars will simply disappear because what has been debited from A and will be available to be seen in account A will the corresponding credit will not be visible.

So, this leads to inconsistent state and to handle that what we need to do is to roll back the transaction, which means that we need to undo the changes that we have already done. So, we have to again go back to account A and write a new value which was the earlier value the value before the debit had happened. And this process of restoring the consistency back to the database is known as the recovery process.

(Refer Slide Time: 03:58)

T_g	T_i
read (A) write (A) read (B)	read (A) commit

- Recoverable schedule
 - If a transaction T_j reads a data item previously written by a transaction T_i , then the commit operation of T_i must appear before the commit operation of T_j
 - The following schedule is not recoverable if T_g commits immediately after the read(A) operation

If T_g should abort, T_g would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

So, we say that a so, let us define a schedule to be recoverable if a transaction T_j reads A data previously written by a transaction T_i , then the commit operation of T_i must appear before the commit operation of T_j , if that happens then that is the earlier transaction which has written the data and T_j the later transaction which is reading the data the

earlier transaction has to commit that is make the changes permanent in the database before T_j actually reads it. If that happens, then we say that that schedule is a recoverable schedule.

So, consider a following schedule of transactions T_8 and T_9 where T_8 has read and written A, but has not committed; that means, some more tasks in T_8 are still pending it has not finished, but T_9 then reads A which is in terms of serializability it is fine, but then T_9 commits and then T_8 is again trying to read B the continues. So, what happens is what if the transaction will fail the transaction T_9 will fail immediately after the read operation.

So, what will happen I am sorry, if T_8 aborts in between, then what will happen that T_9 would have read because, say in read B or of T_8 T_8 aborts that it fails, then T_9 has already read the intermediate value of A and has committed which means it is possibly shown it to the user, but T_8 since it has aborted sent it has failed, it has to be rolled back and the original value of A will be rolled back which is different from what has already been shown to the user and he will reach an inconsistent state.

(Refer Slide Time: 06:01)

The slide features a title 'Cascading Rollbacks' in red at the top right. To the left is a small icon of a sailboat on water. On the far left, vertical text reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr-2018'. The main content area contains a table illustrating transaction scheduling:

T_{10}	T_{11}	T_{12}
read (A) read (B) write (A) abort	read (A) write (A)	read (A)

Below the table, two bullet points explain the cascading effect:

- Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)
- If T_{10} fails, T_{11} and T_{12} must also be rolled back
- Can lead to the undoing of a significant amount of work

At the bottom, there is footer text: 'Database System Concepts - 6th Edition', '33.8', and '©Silberschatz, Korth and Sudarshan'.

So, this is an example of a schedule which is not recoverable. Now let us also observe that a single transaction failure not only means that one transaction needs to be rolled back, but it could have a cascading effect, that is a series of transaction may require a rollback. So, here is an example of T_{10} , T_{11} and T_{12} . So, T_{10} reads A and B and writes A

and then T_{11} reads and writes A and T_{12} reads A and at that time if T_{10} fails if that aborts, then naturally it is not enough to simply roll back T_{10} because, if we roll back T_{10} , then we the value of a goes back to the original and T_{11} would have a wrong value which T_{10} had written, but has now been undone has now been rolled back.

So, it means that T_{11} will also have to be rolled back. Similarly if that is rolled back then naturally T_{12} also have to be rolled back and so on and when this rolling back goes from one transaction to the other we say this is the cascading roll back. And this can lead to a significant amount of work.

(Refer Slide Time: 07:15)

The slide has a title 'Cascadeless Schedules' in red. To the left is a small image of a sailboat on water. On the right is a table showing transaction schedules. The table has three columns labeled T_{10} , T_{11} , and T_{12} . T_{10} has rows for 'read (A)', 'read (B)', 'write (A)', and 'abort'. T_{11} has rows for 'read (A)' and 'write (A)'. T_{12} has a single row for 'read (A)'. The slide also contains a list of bullet points and some footer text.

- **Cascadeless schedules** — for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i appears before the read operation of T_j
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless
- Example of a schedule that is NOT cascadeless

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
abort	read (A) write (A)	read (A)

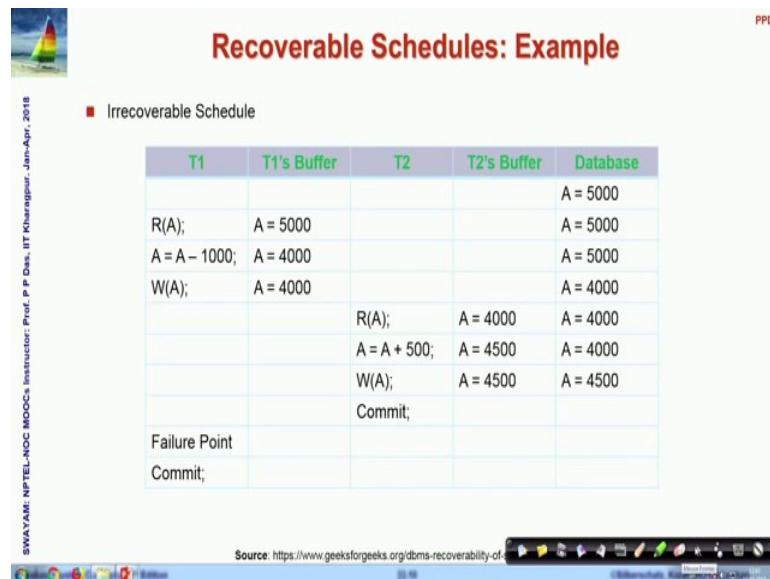
SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr 2018
Database System Concepts - 8th Edition
33.9 ©Silberschatz, Korth and Sudarshan

So, what we would prefer is if we could have schedules where such cascading roll back is not required. So, and there is a there is a condition through which you can achieve that. So, if we have a pair of transaction T_i and T_j . So, that T_j reads A data item previously written by T_i , then the commit operation of T_i has to happen before the read operation of T_j which means that said in other words that T_j should read only read values which are already committed and not read intermediate temporary values of other transactions.

So, every cascadelable schedule is also recoverable because, you can individually recover that and it is desirable to restrict schedules to those which are cascade less as far as possible, we will see that in non not all cases that is possible, but if it is possible you would like schedules which are cascade less. So, that covered a rollback work the extra

work can be minimized. So, here is an example which we had just seen which is not a cascadable schedule.

(Refer Slide Time: 08:25)



The slide has a header 'Recoverable Schedules: Example' and a subtitle 'Irrecoverable Schedule'. It contains a table with columns for T1, T1's Buffer, T2, T2's Buffer, and Database. The table shows the following sequence of operations:

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R(A);	A = 5000			A = 5000
A = A - 1000; A = 4000				A = 5000
W(A); A = 4000				A = 4000
	R(A); A = 4000			A = 4000
	A = A + 500; A = 4500			A = 4000
	W(A); A = 4500			A = 4500
	Commit;			
Failure Point				
Commit;				

Source: <https://www.geeksforgeeks.org/dbms-recoverability-of-transaction-schedule/>

So, wait for word let us take a couple of examples of very similar transactions and, we would see when their schedules are irrecoverable, when their cascaded recovery is possible cascaded rollback is possible and, when cascade less rollback is possible. So, if you so, here what I have done is I have shown here the 2 transactions T_1 and T_2 . And this is what transaction T_1 is doing and we assume that in the database the initial value of a is 5000.

So, what will happen is read here and this value is a different A this is in the buffer or the memory of T_1 transaction, where A becomes 5000, then you subtract 1000 and then you write back the moment you write back in the database in between the value in the database is not changing, it is only that value is only in the buffer and, when you write back the value in the database has changed.

And then transaction T_2 reads that value. So, in its local buffer A becomes 4000 it increments by 500 and then writes it back and when that happens, then in the database also the value has changed to 4500 and then T_2 commits and at this point let us assume that there was if there was a failure. So, this is the point where there was a failure there were other instructions in T_1 as well which is not of our interest right now, and then T_1 would have committed, but what happens if the failure happens at this point naturally the

T_1 needs to roll back T_1 needs to undo this and set the this value 5000 back into the database.

But that would mean that what T_2 has committed T_2 has already committed this value 4500 in the database and therefore, that has been probably been used in other places and shown to the user that will create an inconsistency in the database. So, these are this is a schedule of T_1 and T_2 which cannot be recovered from. So, let us and so what it has what has been violated that T_2 has actually read A value which was in transit and, then it has already committed based on that read value.

(Refer Slide Time: 10:57)

Recoverable Schedules: Example

■ Recoverable Schedule without cascading rollback

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R(A);	A = 5000			A = 5000
A = A - 1000; A = 4000				A = 5000
W(A); A = 4000				A = 4000
Commit;				
		R(A); A = 4000	A = 4000	A = 4000
		A = A + 500; A = 4500	A = 4500	A = 4000
		W(A); A = 4500	A = 4500	A = 4500
		Commit;		

Source: <https://www.geeksforgeeks.org/dbms-recoverability-of-schedule/>

Now, let us look into the next. So, what has been done here that all the changes are the same, but the only point that we have done is we have changed the point where the commit happens again still the T_2 is reading the same value in our in a and is making the updates 4500, but the commit happens at a later point of time after the commit of this transaction T_1 has taken place.

So, this is recoverable, but if we want to recover T_1 naturally; that means, that for T_1 to be recovered, I also need to recover T_2 because T_2 is used a value which is not going to be the value in after the rollback of T_1 has happened T_2 has used 4000, but after the rollback the value in the database will be back to 5000. So, it is the rollback is required for T_1 as well as in T_2 . So, this is a case of cascaded cascading roll back that has

happened. So, some more work is being done and that has happened because T 2 now here the rollback is possible because T 2 is committing after T 1.

So, the transaction it is reading from it is actually committing the changes after that source transaction has committed. So, that satisfies the condition of recoverable schedule. So, you are able to recover, but it still required the cascading because T 1 had read A value in here of A which was not yet committed. So, if we would have committed that, then we would have been able to actually create a schedule which is cascade less as we see in the next slide.

So, now I what the change that has happened is a commit is done, right after writing the value of A and T₂ reads that only after that commit has happened, earlier it was reading before that commit has happened. So, once T₂ reads it after this commit. So, if there is some there is some requirement of if there is some situation of rollback, then only T₁ needs to be rolled back and T₂ does not need to be a rollback because, it has used a value which is already committed.

So, this is the basic through the example you can clearly see, what is how the rollback can happen and in a later module, we will discuss the processes of how to do this kind of rollback the cascading and non cascading both kinds and show how to go ahead with that, but now for now what we learned is schedules need to be recoverable and, preferably cascade less rollback recovery schedules are preferred in case of database transactions.

Now, let us move on and talk little bit about what is available in SQL language in terms of handling transactions.

(Refer Slide Time: 14:12)

The slide has a header 'Transaction Definition in SQL' with a sailboat icon. The main content is a bulleted list:

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction
- In SQL, a transaction begins implicitly
- A transaction in SQL ends by:
 - **Commit work** commits current transaction and begins a new one
 - **Rollback work** causes current transaction to abort
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
 - Implicit commit can be turned off by a database directive
 - ▶ For example in JDBC, connection.setAutoCommit(false);

On the left margin, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is a photo of the speaker, and at the bottom right are navigation icons.

So, SQL we have seen the kind of DDL data definition and data manipulation language paths and those were discussed in terms of our interactive session as well. As a part of data manipulation it is also possible to specify certain specific transaction events. So, a transaction in SQL typically begins implicitly and, it ends by a commit work which says that let us, you commit the current transaction that is make all the changes permanent, in the database make it visible to the user and begin a new work, or it could roll back the transaction which means that all the changes that you had done are rolled back and the transaction basically aborts.

So, in almost all systems by default every SQL statement commits implicitly and, if it has been able to execute successfully, otherwise it rolls back and this implicit commit can be controlled also, it can be in different system there are different ways to control that and say that I do not want implicit commit I would only want commit to be done explicitly.

(Refer Slide Time: 15:22)

Transaction Control Language (TCL)

- The following commands are used to control transactions.
 - **COMMIT** – to save the changes
 - **ROLLBACK** – to roll back the changes
 - **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK
 - **SET TRANSACTION** – Places a name on a transaction
- Transactional control commands are only used with the **DML Commands** such as
 - INSERT, UPDATE and DELETE only
 - They cannot be used while creating tables or dropping them because these operations are automatically committed in the database

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

33.15

©Silberschatz, Korth and Sudarshan

So, for that purpose a part of SQL called the transaction control language has different instructions commit to save the changes roll back to roll back, the changes undo the changes and also to do some do, it in some controlled way by defining savepoint and you can also set the a particular name to a transaction and it is behavior.

So, let us look at examples for doing that soon and these TCL commands are used with specific DML commands they are meaningful in terms of insert update and delete only for example, if you are creating a database or you are doing a select to data retrieval, then these instructions have no role in those transactions.

(Refer Slide Time: 16:09)

TCL: COMMIT Command

PPD

■ The COMMIT is the transactional command used to save changes invoked by a transaction to the database
■ The COMMIT saves all the transactions to the database since the last COMMIT or ROLLBACK command
■ The syntax for the COMMIT command is as follows:
 ● SQL> DELETE FROM Customers WHERE AGE = 25;
 ● SQL> COMMIT;

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
3	kaushik	23	Kota	2000
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

Before DELETE After DELETE

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Des., IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition

33.16

©Silberschatz, Korth and Sudarshan

So, **COMMIT** is a transaction command which is used to save changes and make them permanent based on what has been invoked. So, here you see the example of a customer database and, what I am showing is if you this is the initial state of that table and, before any value has been deleted and if you do select star from customers these 7 records is what you get to see, in view of that you do a delete and then you commit the delete.

So, we say that I have deleted and make that deletion permanent. So, deleting based on age. So, this record is supposed to be get deleted and this record is supposed to get deleted and, after I have done the commit then again if I do the same data retrieval. And now I get to see 5 records only the 2 record number 2 and record number 4 have been permanently deleted. So, this is the way you can explicitly do commit and make the changes permanent.

(Refer Slide Time: 17:11)

TCL: ROLLBACK Command

- The ROLLBACK is the command used to undo transactions that have not already been saved to the database
- This can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued
- The syntax for a ROLLBACK command is as follows:
 - SQL> DELETE FROM Customers WHERE AGE = 25;
 - SQL> ROLLBACK;

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

Before DELETE

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

After DELETE

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

PPD

In terms of rollback it is a command which is used to undo transactions that is the changes that have already not been saved to the database you can roll back.

So, you can roll back or undo transactions only back up in history up to the last commit, or the last rollback command was issued on this. So, again looking at the same example this is the initial state and, then you did a delete as we did last time. So, these 2 records are to be deleted, but then instead of commit we have given a rollback. So, as you give rollback these deletion operations get undone. So, these two records are again back to the table and so, after the rollback if I again do the select I will get to see the 2 records back in my list. So, this is the purpose of the rollback command.

(Refer Slide Time: 18:09)

The slide has a header 'TCL: SAVEPOINT / ROLLBACK Command' with a small logo of a sailboat on the left. On the right, there is a small 'PPD' watermark. The main content is divided into two columns:

- SAVEPOINT:**
 - A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction
 - The syntax for a SAVEPOINT command is:
 - SAVEPOINT SAVEPOINT_NAME;
 - This command serves only in the creation of a SAVEPOINT among all the transactional statements.
 - The ROLLBACK command is used to undo a group of transactions
 - The syntax for rolling back to a SAVEPOINT is:
 - ROLLBACK TO SAVEPOINT_NAME;
- Example:**
 - SQL> SAVEPOINT SP1;
Savepoint created.
 - SQL> DELETE FROM Customers WHERE ID=1;
1 row deleted.
 - SQL> SAVEPOINT SP2;
Savepoint created.
 - SQL> DELETE FROM Customers WHERE ID=2;
1 row deleted.
 - SQL> SAVEPOINT SP3;
Savepoint created.
 - SQL> DELETE FROM Customers WHERE ID=3;
1 row deleted.

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

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

Prof. P. Das is shown in a video thumbnail on the left side of the slide.

Now, you can see transactions often could be long. So, within the transaction you may want to mark certain points. So, that in case you roll back or you need to roll back, you can roll back to that particular point and those points are in the transaction are known as the **SAVEPOINT**. So, this is the format use a save point and give it a name and, then later on you can use those save points for your purpose of rollback.

So, you are again if you are doing a rollback, then you instead of just doing rollback, you now use the save point ID that you had used in naming that particular point up to which you want to roll back and, do a rollback and that will happen only up to that point. So, let us look at an example so, here it is a series of instructions in a DML transaction. So, I initially set SP one as a save point that is I may want to roll back to the beginning, when I delete one record say ID 1. So, 1 record gets deleted, then I again save another save point another save point SP 2 this was SP 1 and, then delete a second record another save point delete another record.

So, now I have a control to undo at this point have a control to undo to 3 points for example, if I do a rollback to SP 3 I will roll back to this point, where only this record will be deletion of this record will be undone, but the first 2 records will still look show as deleted, but if I roll back to save point SP 2, then 2 records ID 2 and ID 3 that were deleted their deletion will be undone and only 1 deletion will look up. Similarly if I roll back to SP 1, it will show that no deletion as it all happened.

(Refer Slide Time: 20:14)

TCL: SAVEPOINT / ROLLBACK Command

PPD

At the beginning

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT * FROM Customers;

After ROLLBACK

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT * FROM Customers;

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

So, if I do that on the this is the initial state on the left to the initial state of the database 3 records have been deleted and, then I do undo off the first deletion of the first 2 and I roll back to SP 2.

So, then when I undo the deletion of the last 2 records, then the what I see is the records which are marked as ID 2 and ID 3, which were done after SP 2 was marked which were deleted after SP 2 are marked, they are back into the table whereas, the deletion of SP 1 is still in effect and therefore, deletion that was none after SP 1 that is of record ID 1 is still missing and in this way you can control and roll back to any specific point in a database in a database transaction.

(Refer Slide Time: 21:13)

The slide is titled "TCL: RELEASE SAVEPOINT Command". It features a list of bullet points:

- The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created
- The syntax for a RELEASE SAVEPOINT command is as follows:
 - RELEASE SAVEPOINT SAVEPOINT_NAME;
- Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

Database System Concepts - 8th Edition 33.20 ©Silberschatz, Korth and Sudarshan

You can once you have marked a safe point you can also, release the safe point that is you can choose to forget that safe point. Once a safe point has been released you cannot roll back to that safe point naturally.

(Refer Slide Time: 21:26)

The slide is titled "TCL: SET TRANSACTION Command". It features a list of bullet points:

- The SET TRANSACTION command can be used to initiate a database transaction
- This command is used to specify characteristics for the transaction that follows
 - For example, you can specify a transaction to be read only or read write
- The syntax for a SET TRANSACTION command is as follows:
 - SET TRANSACTION [READ WRITE | READ ONLY];

Source: http://www.tutorialspoint.com/sql/sql_transactions.htm

Database System Concepts - 8th Edition 33.21 ©Silberschatz, Korth and Sudarshan

You can use **SET TRANSACTION** command to initiate a database transaction also and, it is typically used to specify the characteristics of the transaction, particularly if you want to say whether a transaction is a read only transaction or a read write transaction,

then you can do it in this way, you can say set transaction and give a read or write flag read or write or read only flag for that.

Let us quickly take a look at a different form of serializability besides the conflict serializability is called view serializability.

(Refer Slide Time: 21:59)

The slide has a title 'View Serializability' in red at the top right. To the left of the title is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Dham, IIT Kharagpur - Jan-Apr, 2018'. At the bottom of the slide, there is footer text: 'Database System Concepts - 8th Edition', '33.23', and '©Silberschatz, Korth and Sudarshan'.

View Serializability

- Let S and S' be two schedules with the same set of transactions. S and S' are **view equivalent** if the following three conditions are met, for each data item Q ,
 1. If in schedule S , transaction T_i reads the initial value of Q , then in schedule S' also transaction T_i must read the initial value of Q .
 2. If in schedule S transaction T_j executes $\text{read}(Q)$, and that value was produced by transaction T_k (if any), then in schedule S' also transaction T_j must read the value of Q that was produced by the same $\text{write}(Q)$ operation of transaction T_k .
 3. The transaction (if any) that performs the final $\text{write}(Q)$ operation in schedule S must also perform the final $\text{write}(Q)$ operation in schedule S' .
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone

So, in terms of view serializability we again define what is known as when are 2 transaction schedules defined to be view equivalent, earlier you remember we define 2 schedules to be conflict equivalent, now we are defining view equivalent. So, there are 3 conditions the conditions are simple what conditions say is a to try a schedules are view equivalent, if the transaction the initial value that a transaction reads is same in both these schedules, for every transaction the initial value that it reads must be the same between the 2 schedules.

Similarly, the third condition says that the final write that is done, final value that it writes every transaction writes in both the schedules must be the same the same writes should operate. And the second conditions is a read write pair that every transaction when it performs a read on the data item, it must read from the write corresponding write in the other schedule in by the same by the transaction that which did the write.

So, I always initialize start with the same initial values for every data item in both schedules, I always read from the corresponding right in the same schedule in the 2

schedules and, I must write the final in every transaction every data item must be written in the same way in the 2 schedules.

So, this is again and the key balance is based purely on read write alone as is the case of conflict equivalence also.

(Refer Slide Time: 23:39)

The slide has a title 'View Serializability (Cont.)' with a sailboat icon. It contains a table for schedule T₂₇, T₂₈, and T₂₉. The table shows:

T ₂₇	T ₂₈	T ₂₉
read (Q)		write (Q)
write (Q)		write (Q)

Below the table is a list of questions:

- A schedule S is **view serializable** if it is view equivalent to a serial schedule
- Every conflict serializable schedule is also view serializable
- Below is a schedule which is view-serializable but *not* conflict serializable

At the bottom left is the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Dham, IIT Kharagpur, Jan-Apr, 2018'. At the bottom center is 'Database System Concepts - 8th Edition' and '33.24'. At the bottom right is '©Silberschatz, Korth and Sudarshan'.

So, given the definition of view equivalence, we can say schedule is the view serializable, if it is view equivalent to a serial schedule earlier which said that a schedule is conflict serializable, if it is conflict equivalent to a serial schedule. Now we are defining the view serializability, with a little bit of thought you can convince yourself that every conflict serializable schedule is also view serializable, but the reverse is not true.

So, here is a schedule which is view serializable, but it is not conflict serializable, you know this is not conflict serializable because, certainly you cannot make it into a serial schedule make it equivalent to a serial schedule because, you cannot move this right Q above the right Q of T₂₈ or of T₂₉, but you cannot move this either. So, given that but if you in terms of the view equivalence we balance, then you will say that this is equivalent to a serial schedule and what should be the serial schedule; obviously, there are 6 choices because there are 3 schedules.

So, there are 6 possible permutations which give you 6 different serial schedules and if in that so our first condition says that I must read from the same value so; obviously, T_{27} reads the initial value of Q. So, T_{27} has to be the first transaction, if the third condition says that I must do the same right T_{29} does the final write here. So, the in the serial schedule also T_{29} must be the last one. So, T_{28} has to be the middle one.

So, the serial schedule that this is equivalent to is $T_{27} T_{28} T_{29}$ and, the one reads and the other 2 writes and T_{29} performs a final write. So, you can see that this is a this is not a conflict serializable, but this is view serializable and if you note the view serializability moment, you have you view serializability and you may not have conflict serializability, then you must be having certain blind rights, these are called blind rights this is a blind right, in the sense that here you are writing the value of Q in T_{28} without having read it is current or previous value. So, you have just blindly you had just computed some value and you are writing to that.

So, if a schedule is not conflict serializable, but is view serializable it must have performed some blind rights where it has written data without actually reading it. So, this is a weaker form of serializability that is possible.

(Refer Slide Time: 26:20)

The slide has a title 'Test for View Serializability' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- The precedence graph test for conflict serializability cannot be used directly to test for view serializability
 - Extension to test for view serializability has cost exponential in the size of the precedence graph
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems
 - Thus, existence of an efficient algorithm is *extremely unlikely*
- However, practical algorithms that just check some **sufficient conditions** for view serializability can still be used

At the bottom left, there is a small video thumbnail showing a person speaking. The footer contains the text 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6th Edition', '33.25', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

Now the question is similar to conflict serializability, where we saw that it schedule can be conflict serializable, if it is corresponding precedence graph is a cyclic. So, we would

like to extend find out similar test for view serializability, but as it turns out that trying to find out this is exponential in cost in terms of the size of the precedence graph.

So, it has been proved that the of checking, whether a schedule is view serializable is in the class of NP complete problem. So, if you are good in algorithms. So, you will know what NP problems are and when are problems called NP complete, in very simple terms even if you are not familiar with that depth of algorithms, you can simply issue note that if an algorithm is NP complete, then it is extremely unlikely that there exists an efficient algorithm for.

If there exists any kind of polynomial time algorithm, it is extremely unlikely still not it is still an open problem in computer science, whether a tall polynomial algorithm exists for NP complete problems, but it is extremely unlikely that an efficient algorithm will exist, you may have some approximate algorithms which can give sufficiency conditions which can say that well if these conditions are satisfied, then necessarily a schedule is view serializable, but not a sufficient condition are not a necessary condition, that is in other words that there may be some schedules which do not satisfy the sufficient condition, but are still view serializable.

(Refer Slide Time: 27:59)

View Serializability: Example 1

- Check whether the schedule is view serializable or not?
 - S : R2(B); R2(A); R1(A); R3(A); W1(B); W2(B); W3(B);
- Solution:
 - With 3 transactions, total number of schedules possible = $3! = 6$
 - <T1 T2 T3>
 - <T1 T3 T2>
 - <T2 T3 T1>
 - <T2 T1 T3>
 - <T3 T1 T2>
 - <T3 T2 T1>
 - Final update on data items:
 - A :-
 - B : T1 T2 T3
 - Since the final update on B is made by T3, so the transaction T3 must execute after transactions T1 and T2.
 - Therefore, $(T1, T2) \rightarrow T3$. Now, Removing those schedules in which T3 is not executing at last:
 - <T1 T2 T3>
 - <T2 T1 T3>

Source: <http://www.edugrabs.com/how-to-check-for-view-serializability/>

Database System Concepts - 6th Edition 33.26 ©Silberschatz, Korth and Sudarshan

So, using view serializability have certain problems. So, here I have worked out a longer problem in terms of the view serializability to check that. So, it is kind of a brute force algorithm. So, if you see this is the schedule given there are two data items A and B and

there are 3 transactions T_1 T_2 T_3 . Since there are three transactions, then if I want to prove if it is view serializable, then what I will have to do I will have to find a one of the possible serial schedules which is view equivalent to this?

So, first I list out all the serial schedules given 3 transactions, there are 6 serial schedules and then I first start with condition three which is who is doing the last update. So, there are writes are only on B. So, and last of that are being done in all the 3 transactions. So, there is no write on A so, the list of final update on A is empty and for B the order is T_1 T_2 T_3 so, T_3 does the last.

So, it must whatever schedule this whatever serial schedule this given schedule S has to be view equivalent to must have T_3 as the last transaction to execute. So, only these two are the candidates which may be view equivalent to this schedule S.

(Refer Slide Time: 29:35)

View Serializability: Example 1

- Check whether the schedule is view serializable or not?
 - S : R2(B); R2(A); R1(A); R3(A); W1(B); W2(B); W3(B);
- Solution:
 - Initial Read + Which transaction updates after read?
 - A : T2 T1 T3 (initial read)
 - B : T2 (initial read); T1 (update after read)
 - The transaction T2 reads B initially which is updated by T1. So T2 must execute before T1.
 - Hence, $T_2 \rightarrow T_1$. Removing those schedules in which T2 is executing before T1:
 - $<T_2 T_1 T_3>$
 - Write Read Sequence (WR)
 - No need to check here
 - Hence, view equivalent serial schedule is:
 - $T_2 \rightarrow T_1 \rightarrow T_3$

Source: <http://www.edugrabs.com/how-to-check-for-view-serializability>

So, we reduce down and now we have only to decide whether these two any of these two are view equivalent to the given schedule S. So, moving on with that now next we check condition 1 and condition 2 together.

So, condition one checks that they must read the same value in both the schedule. So, we see that these are the reads that are happening on A. So, we see that on A there are reads happening, I am sorry this is these are the three that is reading A. So, it happens in the order of T_2 T_3 T_1 and T_3 .

So, this is what you find and in terms of B we find that transaction 2 reads B and writes it. So, it has to be in that order. So, it reads it does an initial read in terms of T_2 and, then the first right of that read value is happening in the transaction T_1 after the update of the read.

So, that means, that whatever schedule we look for in terms of view equivalence, they must have in that schedule T_1 must follow T_2 . So, T_2 must happen first because it needs to read the initial value and, then that initial value is used by then there is a right on by T_1 . So, T_2 has to come before T_1 so; that means, we are already in terms of only 2 we have seen that there are two possible candidates based on condition 3, it is $T T_1 T_2$.

So, in these two we only can have this one which is satisfying the other conditions and there is no read write sequence. So, we conclude that indeed $T_2 T_1 T_3$ satisfies all the three conditions of initial read write after read and the final write conditions and therefore, this given schedule S is actually view equivalent to a serial schedule and, it is a view serial schedule and can be used safely for the transaction.

(Refer Slide Time: 31:45)

The slide is titled "View Serializability: Example 2". It features a logo of a sailboat on the left and a "PPD" watermark in the top right. The main content area contains the following text:

- Check whether the schedule is Conflict serializable and view serializable or not?
 - S : R1(A); R2(A); R3(A); R4(A); W1(B); W2(B); W3(B); W4(B)
- Solution is given in the next slide (hidden). First try to solve it and then check the solution.

At the bottom, there is a video player showing a person speaking, with the text "SWAYAM: NPTEL-NOC" and "Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr- 2018". The video player also shows the source URL "Source: http://www.edugrabs.com/how-to-check-for-view-serializability", the duration "33:28", and the copyright notice "©Silberschatz, Korth and Sudarshan".

There is another example given here, where there are four transactions R1 R 2 R 3 and R 4 and there are 2 data items A and B and, you have to find out establish whether this is view serializable or not, I am not working out this one this is worked out in the presentation slide, but I will not show it here you are you should first try it out and, then

once you have been able to do it or you are unable to do that, then you check the solution from the presentation slide.

(Refer Slide Time: 32:26)

More Complex Notions of Serializability

The schedule below produces the same outcome as the serial schedule $\langle T_1, T_5 \rangle$, yet is not conflict equivalent or view equivalent to it

T_1	T_5
read (A) $A := A - 50$ write (A)	read (B) $B := B + 10$ write (B)
read (B) $B := B + 50$ write (B)	read (A) $A := A + 10$ write (A)

- If we start with $A = 1000$ and $B = 2000$, the final result is 960 and 2040
- Determining such equivalence requires analysis of operations other than read and write

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

Database System Concepts - 8th Edition

33.32

©Silberschatz, Korth and Sudarshan

There are different other complex motions of serializability also for example, if you look at this particular schedule this actually is a serializable schedule, this is the effect that it produces will be same as the serial schedule of T_1 T_5 , but if you go through the definitions of conflict equivalence and, view equivalence you will be able to show that this schedule is neither conflict conflict serializable nor view serializable, but yet given the particular.

So, if you just look at the read write this is not a serializable schedule in terms of conflict or view equivalence, but given the fact that it actually performs simple add subtract operations on these variables, using the properties of add subtract operations you would be able to you can actually see that this particular schedule actually is a serializable schedule and, you will get whatever initial values you start with the value that you will achieve through this schedule and the value that will achieve with the serial schedule T_1 T_5 are indeed same in every case.

But this is determining this requires the understanding of other instructions other operations, besides the read and write. So, this is just to show you that using the read write model and conflict and view equivalents and the only not the only ways of getting

to serializability there are more complex models, but we will not go into the depth of these complex serializability aspect.

(Refer Slide Time: 33:56)

Module Summary

- With proper planning, a database can be recovered back to a consistent state from inconsistent state in the face of system failures. Such a recovery is done via cascaded or cascadeless rollback
- View Serializability is a weaker serializability system for better concurrency. However, testing for view serializability is NP complete

SWAYAM: NPTEL-NOCs Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

33.33

©Silberschatz, Korth and Sudarshan

So, we have shown that with proper planning, a database can be recovered back to a consistent state from an inconsistent state, in case of system failure.

And this such a recovery can be through cascaded or cascadeless rollback and, we have also introduced a simpler model of serializability in terms of the view serializable, but testing for view serializability is np complete. So, as an effective algorithm it is not that powerful.

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

Lecture – 34
Concurrency Control /1

Welcome to module 34 of Database Management Systems, in this module and the next we will talk about Concurrency Control a very key concept of database transactions.

(Refer Slide Time: 00:28)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists four bullet points under the heading 'Module Recap': '■ Recoverability and Isolation', '■ Transaction Definition in SQL', '■ View Serializability', and '■ Complex Notions of Serializability'. At the bottom, there is footer text: 'Database System Concepts - 8th Edition', '34.2', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative black bar at the bottom with various icons.

So, in the last module we have talked about continuing on the transactions, we had talked about recoverability of databases, how to satisfy the **ACID** properties the basic transaction in SQL and we have introduced a second form of serializability in terms of the view serializability.

(Refer Slide Time: 00:44)

The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. The main content lists two objectives:

- Concurrency Control through design of serializable schedule is difficult in general. Hence we take a look into locking mechanism and Lock-Based Protocols
- We need to understand how locks may be implemented

On the left side, there is vertical text: "SWAYAM: NPTEL-NOC NOCC's Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 8th Edition", "34.3", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

Now, here in this we will talk more on the different aspects of concurrency control because, it is good that if two schedules are given, we can we may try to prove if they conflict serializable, if their view serializable and then we can use them, but doing that in general while the database is in execution is an extremely difficult problem because, who is going to give the who is who will be able to give the all possible different types of transactions that may happen hundreds of them that may be going on in a in the database at any given point of time.

So, how do you prove that or how do you know whether they are conflicts serial, or which of them conflict serializable sets are of view serializable sets and so, on. So, we introduce a different kind of need to have a different kind of mechanism and that is the mechanism of lock that is used.

(Refer Slide Time: 01:40)

The slide has a header 'Module Outline' in red. On the left, there's a small sailboat icon and a vertical sidebar with text: 'SWAYAM: NPTEL-MOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list: '■ Concurrency Control', '■ Lock-Based Protocols', and '■ Implementing Locking'. Below the list is a video frame showing a man in a suit. At the bottom, it says 'Database System Concepts - 8th Edition', '34.4', and '©Silberschatz, Korth and Sudarshan'.

So, we will discuss about those aspects issues and the lock based mechanisms.

(Refer Slide Time: 01:45)

The slide has a header 'Concurrency Control' in red. On the left, there's a small sailboat icon and a vertical sidebar with text: 'SWAYAM: NPTEL-MOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list: '■ A database must provide a mechanism that will ensure that all possible schedules are both:

- Conflict serializable
- Recoverable and preferably cascadeless

', '■ A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency', '■ Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur', '■ Testing a schedule for serializability *after* it has executed is a little too late!

- Tests for serializability help us understand why a concurrency control protocol is correct

', and '■ Goal – to develop concurrency control protocols that will assure serializability'. Below the list is a video frame showing a man in a suit. At the bottom, it says 'Database System Concepts - 8th Edition', '34.6', and '©Silberschatz, Korth and Sudarshan'.

So, a database must provide a mechanism that will ensure all possible schedules are both conflict serializable, that is a basic requirement and are recoverable and preferable in a cascade less manner. So, that is the basic requirements that we have seen.

Naturally if we have everything as serial that will happen by default, but that will have very poor degree of concurrency and very low throughput. So, concurrency control

schemes will trade off the amount of concurrency that is allowed and, the amount of overhead. So, what will I have to be ensured is I should be able to for example, if I say that the schedules are always serial then the overhead of ensuring concurrency is the minimum, but naturally the benefit is also minimum, we get a very poor throughput.

The more we would like to allow for more and more concurrency in the system, but at the same time we will need to have to see what is the overhead of that what is the cost of that what how do we have to ensure those and, naturally as we I have already said testing for a scheduled to be serializable after it has happened is; obviously, too late and the question is beforehand how do I get to know it in a general setting.

So, we need to have a certain protocol through which that, transactions might be written, a protocol through which the transactions must operate so, that we can achieve good concurrency in the system. So, here our objective is to develop concurrency control protocols that will ensure serializability and if possible cascadeless recovery.

(Refer Slide Time: 03:28)

Concurrency Control

- One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item
 - Should a transaction hold a lock on the whole database
 - ↳ Would lead to strictly serial schedules – very poor performance
- The most common method used to implement locking requirement is to allow a transaction to access a data item only if it is currently holding a **lock** on that item

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

Database System Concepts - 8th Edition

34.7

©Silberschatz, Korth and Sudarshan

So, naturally what you do you try to see whenever we have conflict, the basic problem of serializability is conflict that is what are you are you reading the right data and, what happens if you inadvertently make changes in a data that has already been read by someone else and so on. So, the why we need to achieve isolation of the transactions would be to make the accesses as mutually exclusive as possible.

So, naturally one way it could be to do it using locks. So, we by the basic concept of the lock is you say that this data item is in use. So, others should not use it. Now what should be the data item, should it be the whole database, it can be the whole database we say this database is in use other transaction cannot use it, which boils down to saying almost that you have a serial schedule.

So, at any point of time only one transaction can operate on the database naturally your concurrency will be very poor. So, that is not what is what is acceptable. So, we need locking mechanisms, or a mechanism to control exclusivity in terms of holding locks on smaller items possibly at a record level at a value level and so on. So, that gives rise to a whole lot of lock based protocol some of which we are going to discuss.

(Refer Slide Time: 04:53)

The slide has a title 'Lock-Based Protocols' in red. Below the title is a bulleted list of points about locks:

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
 1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction
 2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction
- A transaction can unlock a data item Q by the **unlock(Q)** Instruction
- Lock requests are made to the concurrency-control manager by the programmer
- Transaction can proceed only after request is granted

At the bottom left is a small video thumbnail of a man speaking. At the bottom right is a navigation bar with icons for back, forward, search, etc. The footer contains the text 'SWAYAM: NPTEL-NOCO: Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr- 2018', 'Database System Concepts - 8th Edition', '34.9', and '©Silberschatz, Korth and Sudarshan'.

So, lock is a mechanism to control concurrent access and to start with there are a variety of locks that exist in a database system variety of types, but to start with we are talking about two locking modes one is exclusive mode, which is designated as X and other is shared mode and which is designated as S.

Naturally the exclusive in the exclusive mode, the data item can be read and written both and such a lock is obtained by doing it **lock-X** instruction and in the shared mode the data item can only be read. So, as you can understand why is it exclusive, when you do read write because if two transactions try to write the same item at the same time, then

you do not know what is who has been successful and who is the last right and what is the actual final value they will become indeterminate.

But if I have a value and multiple transactions read that at the same time certainly there is no problem because, all of them necessarily will read the same data. So, that is what is called shared and a shared lock or a shared mode lock can be obtained in terms of the **lock-X** instruction. And transaction when it has a lock it can unlock that by an unlock on the same data item.

So, there is a concurrency control manager to whom the lock requests are made, whether it is a request to grant, or it is a request to release and a transaction can proceed only after the request has been granted.

(Refer Slide Time: 06:26)

The slide is titled "Lock-Based Protocols". It includes a "Lock-compatibility matrix" table:

	S	X
S	true	false
X	false	false

Below the table is a list of rules:

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
 - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted
- Transaction T_i may unlock a data item that it had locked at some earlier point
- Note that a transaction must hold a lock on a data item as long as it accesses that item
- Moreover, it is not necessarily desirable for a transaction to unlock a data item immediately after its final access of that data item, since serializability may not be ensured

So, let us look into finer details this is what is known as a lock compatibility matrix. So, if there are multiple lock modes which is what is expected, then you try to see which locks can be held or operated simultaneously.

So, this is shown in terms of our present assumption that has shared an exclusive lock naturally, it transact two transactions can hold a shared lock simultaneously on the same data item, but all other combinations that is no two transactions can hold a shared and an exclusive, or two exclusive locks on the same data item at the same time. So, which means that two transactions can read a value at the same time, but two transactions

cannot one is reading the value and other is writing, the value is not possible the reverse is also not possible and two transactions writing, the value is not possible those are called said to be the incompatible modes of loss.

So, if the transaction is granted a lock, if it is compatible with the lock that is already held by another transaction, you cannot get an incompatible lock granted to you. And any number of transactions certainly can hold the shared lock and an item, but if any transaction wants to have an exclusive lock on the item, then no other transaction may hold any lock on that item. So, if I want to write that as a transaction I must be the only transaction who is who has to have that exclusive lock I must be the only transaction who is trying to write, but when I want to read many transactions can simultaneously read.

So, if a lock cannot be granted that if I want a lock either a shared lock, or an exclusive lock and if it cannot be granted, then the transaction has to wait till the all incompatible locks have been released and, only then this lock can be requested lock in being granted. And certainly a transaction who is holding a lock on a data item can unlock it at some point, after its purpose of accessing the data item is over and, a transaction must hold a lock on the data item as long as it is accessing the item that is the basic protocol.

So, you must request first get a grant of that lock do the operations that you want and then you unlock, this is a basic process that has to happen, usually it is said that as soon as you are done with the operations of the data item you mean unlock that, you may want to wait for a little longer for the ensuring the serializability these details we will see subsequently.

(Refer Slide Time: 09:14)

Lock-Based Protocols: Example

■ Let A and B be two accounts that are accessed by transactions T1 and T2.

- Transaction T1 transfers \$50 from account B to account A.
- Transaction T2 displays the total amount of money in accounts A and B, that is, the sum $A + B$.
- Suppose that the values of accounts A and B are \$100 and \$200, respectively

T1: lock-X(B); read(B); $B := B - 50;$ write(B); unlock(B); lock-X(A); read(A); $A = A + 50;$ write(A); unlock(A);	T2: lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A + B)
--	---

■ If these transactions are executed serially, either as T1, T2 or the order T2, T1, then transaction T2 will display the value \$300

SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr., 2018
Database System Concepts - 8th Edition
34.11
©Silberschatz, Korth and Sudarshan

So, let us come to an example. So, here are two transactions T_1 and T_2 . So, this is a the two transactions T_1 , T_2 the transaction T_1 does this operation it transfers 50 dollar from account B to account A.

So, it debits B here credits A here. So, it transfers and transaction T_2 displays the total sum of money in the accounts A and B. So, it reads A reads B displays and say initially the transaction initially let us say these accounts have values 100 and 200. So, what this transaction will do it needs to do the transfer. So, it needs to read B debit and write and what it has to do since it has to read it must have a shared lock. Since it has to write it must have a exclusive lock and, if it has got an exclusive lock it will also be able to read that data. So, what it does it performs an exclusive lock.

So, it requests for an exclusive lock and only on getting that it can do this and, when this is over the purpose is over it unlocks B. Similarly to update a it takes an exclusive lock on a updates and, then releases a lock. Transaction T_2 what it does it has to read and display. So, it does not need an exclusive lock it takes the shared lock reads and unlocks, it again takes a shared lock on B reads and unlocks and finally, displays the two data.

Now, if these transactions are executed serially that is T_1 after T_2 or T_2 after T_1 , then the transaction T_2 will always display the value 300 because, 300s is the initial value that we will be able to see if T_2 runs first and T_1 300 is all so, the final value because only 50

dollar has been transferred from B to A. So, the sum remains same. So, you will be able to see that if T_2 runs after T_1 . So, the consistency of the database is maintained.

(Refer Slide Time: 11:32)

	T_1	T_2	concurrency-control manager
	lock-X(B) read(B) $B := B - 50$ write(B) unlock(B)	lock-S(A) read(A) unlock(A) lock-S(B) read(B) unlock(B) display(A + B)	grant-X(B, T_1) grant-S(A, T_2) grant-S(B, T_2) grant-X(A, T_1)
$T_1:$	lock-X(B); read(B); $B := B - 50;$ write(B); unlock(B); lock-X(A); read(A); $A := A + 50;$ write(A); unlock(A);	lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A + B)	
$T_2:$		lock-X(A) read(A) $A := A - 50$ write(A) unlock(A)	

Schedule 1

Now, let us see let us consider a schedule written here as schedule 1 and the the transactions are executing concurrently. So, then this is a possible schedule and let us see what will happen. So, what it does this is where the lock exclusive lock on B is held and B is updated, then A is read then B is read display is done and then this update on a has happened. And this is where we are showing that how the grants are happening.

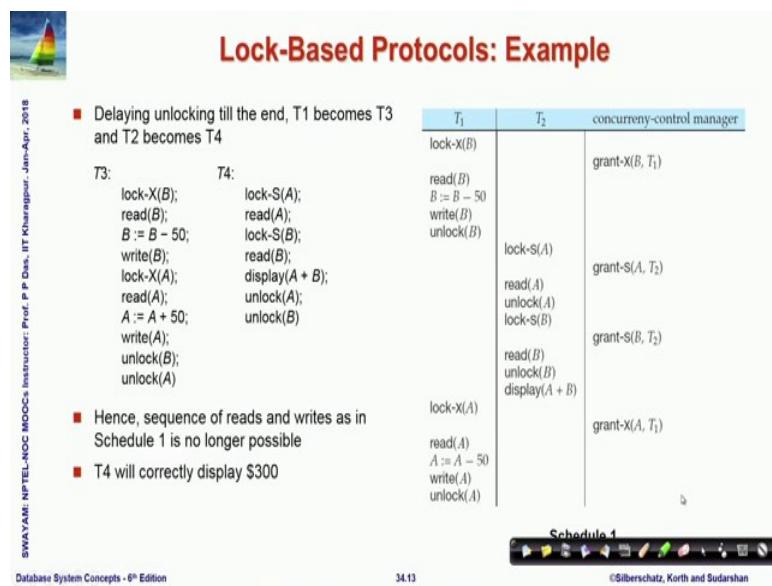
So, as the lock is requested then the request goes to the system that a exclusive lock on B is requested by transaction T_1 . So, that subsequently gets granted and only when the grant has happened the corresponding axis can start, but it can be any indeterminate amount of time between the request of the lock which is here and, the actual grant of the lock, but this operation can happen only after the grant has happened.

So, in every case that is what has to be observed right. Now what happens in this schedule what will be the consequence. So, in this schedule if we look at the transaction T_2 will display only 250 dollar, it will not display 300 dollar why because, if you if you look at this carefully, if you look at this carefully this is where B has got updated. So, B has become 50 dollar less. And then the whole of A and B have been read and displayed. So, naturally the total sum is 50 dollar less.

So, even though we have used a lock it has not been able to achieve the required even though, we have used the lock we have not been able to achieve the required serializability. And it is possible to create a schedule where inconsistent data is getting generated T_2 is actually reading an inconsistent data. So, you have seen inconsistent state in terms of here. Why did it happen? This happened because if we look carefully this has happened because, T_1 has unlocked to prematurely T_1 has unlocked as soon as the update to be was over.

So, it was possible for T_2 to read that value of B which is not what is desirable and we will see that we might want to delay the unlocking till the end, let us see what happens if we do that.

(Refer Slide Time: 14:43)



So, we are here now it is the same transaction in terms of the notion, but T_1 has been made to T_3 here, where you have seen that unlocking is been pushed to the end T_2 has been made into T_4 , where the unlocking is pushed to the end. And now naturally if you look into this, if you wanted to do a schedule 1 you cannot do this kind of a schedule 1 that schedule will not be permissible because, you will not be able to get the locks, T_4 will not be able to get the locks that T_2 could get in the sequence of reads and writes in schedule 1 is no longer possible.

(Refer Slide Time: 15:52)

Lock-Based Protocols: Example

- Given, T₃ and T₄, consider Schedule 2 (partial)
- Since T₃ is holding an exclusive mode lock on B and T₄ is requesting a shared-mode lock on B, T₄ is waiting for T₃ to unlock B
- Similarly, since T₄ is holding a shared-mode lock on A and T₃ is requesting an exclusive-mode lock on A, T₃ is waiting for T₄ to unlock A
- Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution
- This situation is called **deadlock**
- When deadlock occurs, the system must roll back one of the two transactions.
- Once a transaction has been rolled back, the data items that were locked by that transaction are unlocked
- These data items are then available to the other transaction, which can continue with its execution

T ₃	T ₄
lock-X(B); read(B); $B := B - 50;$ write(B);	lock-S(A); read(A); lock-S(B); read(B); lock-X(A); read(A); $A := A + 50;$ write(A); unlock(B); unlock(A)
lock-X(A)	lock-S(A) read(A) lock-S(B)

Schedule 2

SWAYAM: NPTEL-NOC NOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur; Jan-Apr., 2018
Database System Concepts - 6th Edition
34.14
©Silberschatz, Korth and Sudarshan

So, whatever way we actually do the schedule T₄ will always correctly show, that the sum is three hundred dollar. So, here we are again showing T₃ T₄ this is a schedule given schedule 2, which is just given partially. And since T₃ is holding an exclusive lock on B and T₄ is requesting a shared lock. So, if I hold it this is T₃ is holding an exclusive lock and T₄ is requesting for a shared lock.

So, T₄ has to wait for T₃ to unlock B before it can actually do that operation. Similarly you will find if you look further T₄ has already got a shared lock to read A. And T₃ needs a shared lock on I am sorry T₃ needs an exclusive lock on A to be able to proceed. So, this one is here. So, T₄ cannot go beyond this point because T₃ has the lock on B and, one is this T₃ cannot go beyond this point because T₄ has that shared lock.

So, what we situation are we getting into. So, we are getting into a situation where, neither of T₃ or T₄ can actually proceed the normal execution, T₃ is waiting for exclusive lock on A and which T₄ has and T₄ is waiting for the shared lock on B, which T₃ already holds as an exclusive manner. And this in so, this is kind this is what is called deadlock, if you have a studied operating system, then you have must be knowing deadlock very well, and there the deadlock happens to different other issues of sharing resources here it is because of the lock.

So, moment you use locks there is a possible danger of having deadlock. And once you have a deadlock there is no other way than to unroll one or more of the transaction, then start all over again, it has to roll back one of the two transactions to be able to proceed. And once the transactions are rolled back the data items that were locked by the transactions will also be unlocked. So, please understand this in view of the earlier discussion we had in terms of transact TCL commands.

So, when you actually which we are roll back we had at that point could only say that your value of the data item in the database will be rolled back, but certainly as now you can understand that, if you roll back also the locks that you have required we also get unlocked so, that other transactions can get those locks and proceed. So, then the data items become available for other transactions and, that can continue the execution.

(Refer Slide Time: 19:03)

The slide has a title 'Lock-Based Protocols' in red. To the left is a small logo of a sailboat on water. On the right is a decorative footer bar with icons. The main content is a bulleted list of nine points about locking and deadlocks.

■ If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states
■ On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur
■ Deadlocks are a necessary evil associated with locking, if we want to avoid inconsistent states
■ Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back transactions, whereas inconsistent states may lead to real-world problems that cannot be handled by the database system
■ A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks
■ Locking protocols restrict the set of possible schedules
■ The set of all such schedules is a proper subset of all possible serializable schedules
■ We present locking protocols that allow only conflict-serializable schedules, and thereby ensure isolation

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

Database System Concepts - 8th Edition

34.15

©Silberschatz, Korth and Sudarshan

So, if we do not so, so we are saying that we wanted to use locks to get better control on the serializability and so on. And it was partly possible, but then we are getting into different other kinds of different problems.

So, if you do not use locking or, if we unlock data items very early then after reading, or writing them then we may get inconsistent state this is what you have seen, on the other hand, if we do not unlock a data item before requesting a lock on another data item that is if we hold it on for a very long time, then deadlock may occur. So, if we do it too soon

we do not use the lock that we have a problem of inconsistent state, if we do it hold it for too long then there could be problem of deadlock.

Now, deadlocks are necessarily evil of locking, if you do locking you will always face a deadlock, if we want to avoid inconsistent states. Now between these two; obviously, we would prefer deadlock the reason, we will prefer deadlock to inconsistent state is the fact that, if we have deadlock we still have the option of rolling back and we can take different strategies to decide what to rollback and how much to rollback and so on whereas, inconsistent states may lead to real world problems that cannot be handled by the database system.

In fact, in some in many cases I may get into some inconsistent state which is very difficult to even recognize that it is an inconsistent state. So, we will continue and prefer deadlocks over inconsistent states and, we will define we will try to define different locking protocols a set of rules that the transactions should follow, while the request and release locks to make our life relatively easier.

So, locking protocols necessarily will restrict the set of possible schedules because, we will put in some discipline in terms of how we look and how we release them, and the set of all such schedules is a proper subset of possible serializable schedules that is easy to understand. And we will present locking protocols that allow only conflict serializable schedule which ensures isolation.

(Refer Slide Time: 21:23)

The Two-Phase Locking Protocol

- This protocol ensures conflict-serializable schedules
- Phase 1: Growing Phase
 - Transaction may obtain locks
 - Transaction may not release locks
- Phase 2: Shrinking Phase
 - Transaction may release locks
 - Transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points**
 - That is, the point where a transaction acquired its final lock

SWAYAM: NPTEL-NOCOCS Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8th Edition

34.16

©Silberschatz, Korth and Sudarshan

So, let us look at the most widely used protocol this is called the two phase locking protocol which guarantees conflict serializability, it does a simple thing it has two phases a growing phase, where it transaction may obtain locks and may not release any lock. And a shrinking phase which the transaction may release locks and may not obtain any law. So, you are just separating out the you know the grant or the access of locks holding of locks and the releasing of locks into two different phases you do not mix them up.

And that is the two phrases phases of the locking protocol. And this ensures so, we are we will not do the proof, but you can look it up in the book or, but you can see through examples that it can be shown that transactions can be serialized in the order of the points where they do the locking. So, these are known as lock points and, that is where the transaction actually acquired it is final lock.

(Refer Slide Time: 22:25)

The Two-Phase Locking Protocol (Cont.)

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
 - Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

34.17

©Silberschatz, Korth and Sudarshan

So, there can be conflict serializable schedules that cannot be obtained, if two phase locking is used. So, what this is saying if you use two phase locking you are guaranteed to have conflict serializable schedule, but there are conflicts serializable schedules for which you may not be able to honor the two phase locking protocol. So, two phase locking protocol is kind of a sufficiency condition.

(Refer Slide Time: 22:50)

Lock Conversions

- Two-phase locking with lock conversions:
 - First Phase:
 - can acquire a lock-S on item
 - can acquire a lock-X on item
 - can convert a lock-S to a lock-X (upgrade)
 - Second Phase:
 - can release a lock-S
 - can release a lock-X
 - can convert a lock-X to a lock-S (downgrade)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

34.18

©Silberschatz, Korth and Sudarshan

So, you could also refine the two phase locking with what is known as lock conversion that is you can acquire in the growing phase, or the first phase you can acquire a

exclusive lock, a shared lock, or you can convert a shared lock that you already have into an exclusive lock which is called the lock upgrade process.

Similarly, in the shrinking phase you can release a shared lock release an exclusive lock, or you are holding an exclusive lock you can make it a shared lock. So, you can download. So, you can understand that upgrade and downgrade are strategies to only use that much of restriction that you need, to impose on others and to allow others to access the data to the based possible way. This protocol again issuers serializability and the it certainly depends on the programmer as to how the programmer inserts the various locking instructions.

(Refer Slide Time: 23:46)

The slide has a title 'Automatic Acquisition of Locks: Read' in red. On the left is a small sailboat icon. The main content is a pseudocode algorithm:

```
■ A transaction  $T_i$  issues the standard read/write instruction, without explicit locking calls  
■ The operation  $\text{read}(D)$  is processed as:  
    if  $T_i$  has a lock on  $D$   
        then  
            read( $D$ )  
        else begin  
            if necessary wait until no other transaction has a lock-X on  $D$   
            grant  $T_i$  a lock-S on  $D$ ;  
            read( $D$ )  
        end
```

At the bottom left is a photo of a man, and at the bottom right is a navigation bar with icons.

Now, you will have to when you want to do read or write, you may acquire locks automatically the database systems will allow that. So, this is a very simple algorithm. So, if you want to read a data, I if you have a lock already on that either shared, or exclusive you can simply read it, if you do not have that then if you may have to wait until no other transaction has an exclusive lock on that because, you know that read or shared lock is not compatible with the exclusive lock.

So, you may have to wait till all are the in no other transaction the transaction that was having exclusive lock possibly has released it and, then take a grant of the shared lock on this item and, then read it is a very simple algorithm to automatically acquire locks.

(Refer Slide Time: 24:37)



Automatic Acquisition of Locks: Write

■ `write(D)` is processed as:

```
if  $T_i$  has a lock-X on D
  then
    write(D)
  else begin
    if necessary wait until no other transaction has any lock on D,
    if  $T_j$  has a lock-S on D
      then
        upgrade lock on D to lock-X
      else
        grant  $T_i$  a lock-X on D
    write(D)
  end;
■ All locks are released after commit or abort
```

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

Database System Concepts - 8th Edition

34.20

©Silberschatz, Korth and Sudarshan

Write is little bit more complex because to be able to write either, you already have an exclusive lock on D, then you write or you may have to wait till no other transaction has any log because, exclusive lock is not compatible with shared lock or with other exclusive lock.

So, as long as some transaction has a lock on D you cannot proceed, but once you come to a state, that you already if no other transaction has a lock, then you see whether you yourself have a shared lock on D, if you have a shared lock then you upgrade it to an exclusive lock, if you do not have a shared lock, then you they take a grant of the exclusive lock and then you can go and write. So, it is if you follow the two phases these algorithms become very simple. And when you commit or the abort the transaction, then naturally all locks are get will get released.

(Refer Slide Time: 25:32)



Deadlocks

■ Two-phase locking does not ensure freedom from deadlocks

	T_3	T_4
lock-X(B);	lock-S(A);	lock-x (B)
read(B);	read(A);	read (B)
$B := B - 50;$	lock-S(B);	$B := B - 50$
write(B);	read(B);	write (B)
lock-X(A);	display(A + B);	
read(A);	unlock(A);	lock-s (A)
$A := A + 50;$	unlock(B);	read (A)
write(A);		lock-s (B)
unlock(B);		
unlock(A);		

■ Observe that transactions T_3 and T_4 are two phase, but, in deadlock



SWAYAM: NPTEL-HOC MOOCs Instructor: Prof. P P Dhas, IIT Kharagpur - Jan-Apr, 2018
Database System Concepts - 8th Edition
34.21 ©Silberschatz, Korth and Sudarshan

So, the two phase protocol we have already seen that does not ensure freedom from deadlock, you can may follow two phase locking protocol here is an example, but you may still have schedules which will have deadlocks. So, this is one example you can just convince yourself.

(Refer Slide Time: 25:52)



Starvation

■ In addition to deadlocks, there is a possibility of **starvation**

■ **Starvation** occurs if the concurrency control manager is badly designed. For example:

- A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item
- The same transaction is repeatedly rolled back due to deadlocks

■ Concurrency control manager can be designed to prevent starvation



SWAYAM: NPTEL-HOC MOOCs Instructor: Prof. P P Dhas, IIT Kharagpur - Jan-Apr, 2018
Database System Concepts - 8th Edition
34.22 ©Silberschatz, Korth and Sudarshan

There is another problem that can happen, in addition to deadlock this is a code there is a possibility of what is known as starvation; starvation, occurs usually it occurs when the control concurrency control manager is not a efficient one.

So, what did we see in terms of automatic locks in read and write operation is you may have to wait because, someone else is holding a lock on an item. Now holding an exclusive lock on the item, now it is possible that like the current transaction there may be couple of other transactions who are also waiting for a lock on that item and, when the opportunity comes that there is no log being held by any transaction, one of the waiting transactions must be given the lock you cannot if it is an exclusive lock you cannot give it to more than one transaction, but say three transactions were waiting for the exclusive lock and one of them get, that and that transaction can proceed the other transactions have to rollback because, they are not getting the lock.

So, now you again start you again come to the point where you wanted the exclusive lock on that item and at that time somebody is holding it and there are other transactions who are also requesting for exclusive lock. And when you come back and when finally, the exclusive lock is released by all other transactions, then again it is possible that while you are waiting some other transaction that was waiting who gets that exclusive lock and you do not get that so, you roll back and this could repeatedly could keep on happening.

So, if you have a weak strategy in terms of concurrency control, you have you will see that you have had infinite possibilities infinite occurrences, where you could have got that exclusive lock, but you are not being able to get that and therefore, you starve on the data and this is known as a data starvation problem which will also have to be checked while we do the concurrency control policies.

(Refer Slide Time: 27:57)

The slide has a header 'Cascading roll-back' with a sailboat icon. On the left, there is a list of bullet points. On the right, there is a table showing transaction schedules for T₅, T₆, and T₇.

List of points:

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil
- When a deadlock occurs there is a possibility of cascading roll-backs
- Cascading roll-back is possible under two-phase locking
- In the schedule here, each transaction observes the two-phase locking protocol, but the failure of T₅ after the read(A) step of T₇ leads to cascading rollback of T₆ and T₇.

Table:

T ₅	T ₆	T ₇
lock-X(A) read(A) lock-S(B) read(B) write(A) unlock(A)		
	lock-X(A) read(A) write(A) unlock(A)	lock-S(A) read(A)

SWAYAM: NPTEL-NOC INOCCS Instructor: Prof. P. P. Dabholkar, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
34.23
©Silberschatz, Korth and Sudarshan

There is a the potential for deadlock exists in most locking protocols, as we have seen and when a deadlock occurs there is a possibility of cascading roll back because, when it deadlock happens then naturally you will have to roll back. So, you may have to do a cascading roll back as this example is showing. And it is possible for a two phase locking protocol have we have in the example is shown here, where all the transactions are following cascading roll back has to as following two phase locking protocol, but if T₅ fails after the read step of T₇ after the read step of T₇, if T₅ fails then it leads to a cascading rollback T₇ T₅ has to be rolled back. So, T₆ will have to be rolled back, so T₇ will have to be rolled back and so on. ah

(Refer Slide Time: 28:54)

The slide has a header 'More Two Phase Locking Protocols' with a sailboat icon. The content includes two bullet points:

- To avoid Cascading roll-back, follow a modified protocol called **strict two-phase locking**
 - a transaction must hold all its exclusive locks till it commits/aborts
- **Rigorous two-phase locking** is even stricter.
 - All locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit

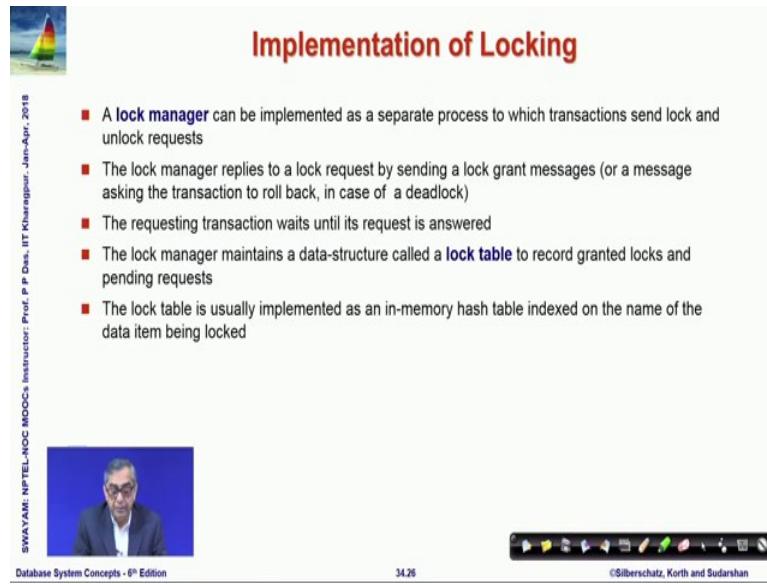
On the left margin, vertical text reads: SWAYAM, NPTEL-NOC INOC-C Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018

At the bottom, there is a video frame showing a man speaking, the text 'Database System Concepts - 8th Edition', the number '34.24', and a navigation bar.

Interestingly there are several other protocols and particularly two more two phase locking protocol 1 is called strict two phase locking, which avoids cascading roll back, where a transaction must hold all exclusive locks till it finally, commits and aborts naturally you can figure out that you are making the time for the transaction to hold lock longer. So, naturally the level of concurrency will go down that is all possible serializable schedules will be smaller, but this guarantees that you will not have a cascading roll back. And there is an even stricter rigorous two phase locking where all locks are held till commit or abort.

In the strict 1 only exclusive locks are held till commit or abort there is a till the end of that transaction, but in rigorous two phase locking all locks are held till the committed abort, in this protocol transaction can be serialized, in the order in which they do the commit and in that way this is a serializable protocol, which also avoids the cascading roll back up. Now finally, before you close this module a quick word in terms of how do you implement locking.

(Refer Slide Time: 30:09)



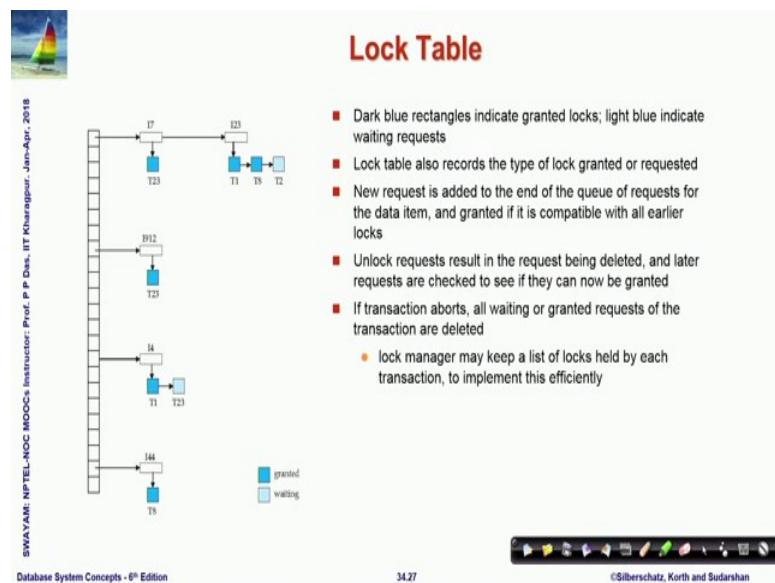
The slide has a title 'Implementation of Locking' in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of five points about lock managers:

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked

At the bottom left, it says 'SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Dabholkar, IIT Kharagpur - Jun-Apr., 2018'. In the center, there is a video frame showing a man speaking. At the bottom right, there is a navigation bar with icons and the text 'Database System Concepts - 8th Edition', '34.26', and '©Silberschatz, Korth and Sudarshan'.

It is the lock there is a lock manager, which implements the locking the lock manager itself runs on a different process to which every transactions end lock and unlock requests. And the lock manager maintains a data structure to maintain what are the transactions, who are holding different locks on different items and based on that the grant messages are queued on that data structure and, these messages actually release the locks and, otherwise the transaction has to wait the lock manager maintains this as a lock table. And this is typically a in memory hash table because, it needs to naturally be very fast and is in the name of the data item being locked.

(Refer Slide Time: 31:01)



So, let us just show you and so, these are the different this is an instance of a lock table and, the nodes are different data items. So, I_7 , I_9 , I_{23} , I_4 , I_{44} , I_{23} are different data items this is a hash table. So, you can see that on I_7 and I_{23} there is a collision and there is collate state chain happening on that. And then for every item you have you maintain a list of locks that are granted to different transactions and the list of requests that are waiting, the dark blue here shows the grant and the light blue shows a waiting status here.

So, it takes it naturally says what type of lock is granted and requested and based on this therefore, when you get a request to put it in the you come and put it you hash it to that data item, put that request on that queue and based on the current status you can decide, whether it can be granted or it has to wait. So, it is added new requests are added at the end of that queue, it is first in first out and whenever a release happens, then naturally a granted node is removed and a waiting node might get a chance to block that item, if the transaction reports all waiting, or granted requests of the transactions certainly will get deleted ok.

(Refer Slide Time: 32:30)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains two bullet points:

- Understood the locking mechanism and protocols
- Realized that deadlock is a peril of locking and needs to be handled through rollback

On the far left, vertical text reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 8th Edition". The bottom right corner features the copyright notice "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is at the very bottom.

So, this is a simple way to manage the locks. So, in this module on concurrency control we have understood the basic locking mechanism and protocols, we have specifically looked at the lock compatibility matrix and the strategies of granting and releasing locks and, we have seen the consequent danger of having deadlock and in some cases starvation, which we have agreed to live with. So, if deadlock happens we will have to roll back one or more transactions and then restart again and, but we cannot take the risk of not having serializable transactions because, that might lead to inconsistent state of the database which is not acceptable.

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

Lecture – 35
Concurrency Control/2

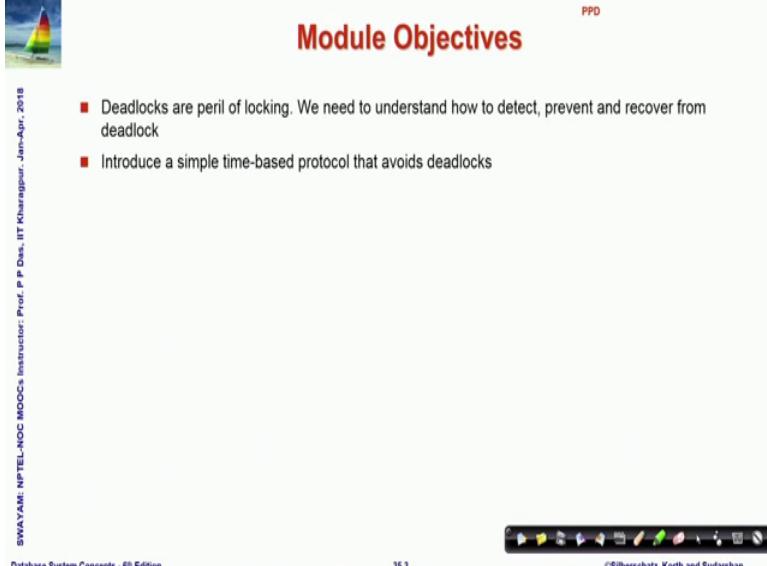
Welcome to module 35 of Database Management Systems. We have been discussing about concurrency control, this is a second and concluding module on that.

(Refer Slide Time: 00:29)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. The footer contains copyright information: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. © 2018', 'Database System Concepts - 8th Edition', '35.2', and '©Silberschatz, Korth and Sudarshan' along with a navigation bar.

So, in the last module, we have talked about the basic issues in concurrency control; and particularly talked about lock based protocol and how to implement locking in very simple terms.

(Refer Slide Time: 00:39)



The slide has a header "Module Objectives" in red. On the left, there is a small sailboat icon and some vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". On the right, it says "PPD". Below the header is a bulleted list:

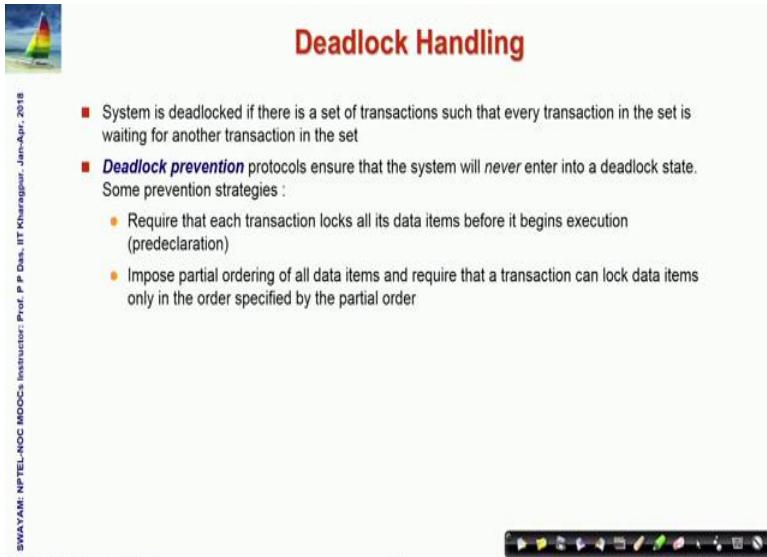
- Deadlocks are peril of locking. We need to understand how to detect, prevent and recover from deadlock
- Introduce a simple time-based protocol that avoids deadlocks

At the bottom, it shows "Database System Concepts - 8th Edition", "35.3", and "©Silberschatz, Korth and Sudarshan". There is also a decorative footer bar with various icons.

As we have seen that deadlocks of the perils of locking I mean we cannot do without locking and certainly if we lock then deadlocks are inevitable almost to happen. So, here first we try to understand how since dead locks are inevitable.

So, there has to be mechanisms to detect deadlocks and recover from them. And also we would like to look at if it is possible to create strategies which can prevent deadlock from happening at all. And so after having studied that we would like to understand take a look into a simple time-based protocol that can avoid deadlock.

(Refer Slide Time: 01:27)



The slide has a header "Deadlock Handling" in red. On the left, there is a small sailboat icon and some vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". Below the header is a bulleted list:

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set
- **Deadlock prevention** protocols ensure that the system will never enter into a deadlock state. Some prevention strategies :
 - Require that each transaction locks all its data items before it begins execution (predeclaration)
 - Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order

At the bottom, it shows "Database System Concepts - 8th Edition", "35.6", and "©Silberschatz, Korth and Sudarshan". There is also a decorative footer bar with various icons.

So, deadlock handling. So, system is deadlock if there is the again just to recap the simple idea is if there is a set of instructions such that every transaction in the set is waiting for another transaction in the set and therefore none of them can actually proceed. So, deadlock prevention protocol ensures that the system will never enter into the deadlock state.

So, the question is can we make some strategy. So, why are we getting into the deadlock, because transactions are making requests for different locks and those are granted. And then some more requests come and we come to a state where A is waiting for B, B is waiting for C, C is waiting for a kind of a situation and we get into a deadlock.

So, can we have strategies so that the requests and releases are done in a way, so that the deadlock will not happen at all. So, I mean fortunately such number of such strategies exist. For example, one strategy which is called a pre-declaration which required that each transaction locks all debt items before it begins its execution that can be shown that that ensures that you will never have deadlock because where in very simple terms you will not be able to start before you have got all the locks.

And once you have got all the locks naturally you have every access to all possible data items and therefore, you will be able to proceed. Naturally, the flip side of this is this will delay the beginning of the transactions to a great extent in many cases, and particularly will bring down the level of concurrency that you can have.

The other which is smarter is what it does is imposes a kind of partial ordering of all the data items that a transaction and all the data items that exist. And it requires that the transaction can lock the items in only in that specific order. So, the important thing here is a partial order among the data items.

And the fact that you locked data items in that order that is specified by the partial order, you cannot lock out of order. And if you can do that then it can be shown that the deadlock will get prevented. We cannot we do not have time to go into the details of how that works, but I just want you to know that such strategies of prevention exist.

(Refer Slide Time: 03:47)

The slide has a title 'Deadlock Prevention' in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list under two sections: 'wait-die scheme — non-preemptive' and 'wound-wait scheme — preemptive'. The 'wait-die' section includes points about older transactions waiting for younger ones, younger transactions never waiting for older ones, and a transaction dying multiple times. The 'wound-wait' section includes points about older transactions forcing rollbacks of younger ones instead of waiting, younger transactions waiting for older ones, and fewer rollbacks than the 'wait-die' scheme. The footer contains the text 'SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '35.7', and '©Silberschatz, Korth and Sudarshan'.

So, the other possible prevention schemes that we will we would like to look at little bit more depth is the fact that I can use timestamps for the transaction. And use those timestamps that is which will tell me which is an earlier transaction in which, a later transaction for preventing deadlock and several strategies for that exist. We will discuss two of them. First is what is known as wait and die with scheme, which is non-preemptive. Non-preemptive means that well in this no one preempts anyone else.

So, what do you do in wait and die, the older transactions because you assume that every transaction has a timestamp. So, smaller the timestamp, older is a transaction. So, older transaction may wait for the younger one to release the item. So, if two transactions are conflicting then the older one will wait; and the younger transaction will never wait for the older one, they are rolled back instead.

So, if there is a conflict, then the younger one in that will always roll back, and the older one will wait. So, a transaction may die several times before acquiring the needed data item kind of starvation may happen, but certainly there will not be a deadlock. Because my A waiting on B, and B waiting on A, cannot happen because out of A and B one must be older has to be older, and that only will wait, the other one will abort, abort and roll back on.

The other is a preemptive scheme where which is called wound and wait scheme, where the older transaction wounds up or forces a rollback of the younger transaction instead of waiting for it that is why this is preemptive. So, the older transaction is preempting the young that transaction to continue to wait and forces it to roll back to abort and that, but the younger transaction may wait for the older one.

So, by doing this preemptive one also it is possible to have a fewer roll backs than the other scheme. So, it is a preemptive scheme, but it the advantages it might allow you fewer roll backs to happen. And with these two kind of timestamp based schemes it is possible to actually prevent deadlocks and for that reason these kind of schemes are often preferred in many context.

(Refer Slide Time: 06:21)

The slide has a title 'Deadlock Prevention' in red at the top right. On the left is a small sailboat icon. The main content area contains two bullet points:

- Both in *wait-die* and in *wound-wait* schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided
- **Timeout-Based Schemes:**
 - a transaction waits for a lock only for a specified amount of time. If the lock has not been granted within that time, the transaction is rolled back and restarted,
 - Thus, deadlocks are not possible
 - simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. At the bottom center: 'Database System Concepts - 6th Edition' and '35.8'. At the bottom right: '©Silberschatz, Korth and Sudarshan'.

So, both in wait and die, and wound and wait scheme, the rollback transaction is restarted with its original timestamp. This is a very very important point to note. When you restart, so your rollback so you have to restart that transaction you restart the transaction you do not put the timestamp of when it is being restarted, you put the timestamp of its original time; The time when it was started and had to be aborted and rollback.

So, the older transactions have precedence over the newer ones and that starvation will get avoided.

So, now what becomes you are you are actually a new candidate because you have been rolled back in and started again, but you carry your older timestamp. So, your precedence has gone higher because in wait and die, and wound and wait in both actually the older one has a precedence.

So, by carrying your older timestamp, you inherently bring in a higher precedence in the system. And in this way there is a precedence based ordering that will naturally always happen. So, this will not only avoid deadlock, but this will also ensure that starvation is avoided; So, very simple and nice scheme.

So, in this you usually have time out basically my transaction waits for a lock only for a specified amount of time. If the lock has not been granted within that time the transaction is rolled back and restarted, and therefore, the deadlock is not possible. It is simple to implement, but starvation can happen in the timeout based scheme. And it is also difficult to determine what is a good time interval to wait.

If you wait too short, then you will spend a lot of time in the in the rollback and restart. If you wait for too long, then your throughput will go down because several transactions are basically waiting on logs. So, theoretically it does avoid deadlock, but in terms of starvation and in terms of the practicality, this there are critical things to decide on this.

(Refer Slide Time: 08:24)

The slide has a header 'Deadlock Detection' with a sailboat icon. The main content is a bulleted list of points about deadlock detection:

- Deadlocks can be described as a *wait-for graph*, which consists of a pair $G = (V,E)$,
 - V is a set of vertices (all the transactions in the system)
 - E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.
- If $T_i \rightarrow T_j$ is in E , then there is a directed edge from T_i to T_j , implying that T_i is waiting for T_j to release a data item
- When T_i requests a data item currently being held by T_j , then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when T_j is no longer holding a data item needed by T_i
- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles

Navigation icons and page numbers are at the bottom.

The second issue in terms of deadlock that we must be able to answer is well hundreds of transactions are going on in the system. Now, how do you know that a deadlock has happened? Because if a deadlock has happened and if you are not using a preventive scheme to ensure that the deadlock will not will never happen theoretical proof; If you are allowing say two phases locking kind of protocol where deadlocks can happen then you must know what the must be able to detect that a deadlock has happened and then take care of it to rollback the transaction.

So, for doing this, we again create a graph, which is wait for graph which is very similar to the precedence graph we saw earlier which the nodes are the transactions and the edges are ordered pair of transactions. So, what do you put an edge from T_i to T_j , you put this edge in what it means is T_i is waiting for T_j . So, if we have a conflict, then certainly one transaction is holding the lock and other is other has requested for that lock.

So, what you do you put an edge for from the one that is waiting for the lock to the one that is already holding the lock for the release of the lock, and in this way the graph gets built up. So, naturally when T_i requested it item currently being held by T_j , then the edge T_i-T_j is inserted in the in this graph. And when the release happens then this edge is removed because T_j is no more holding the item that T_i had actually required.

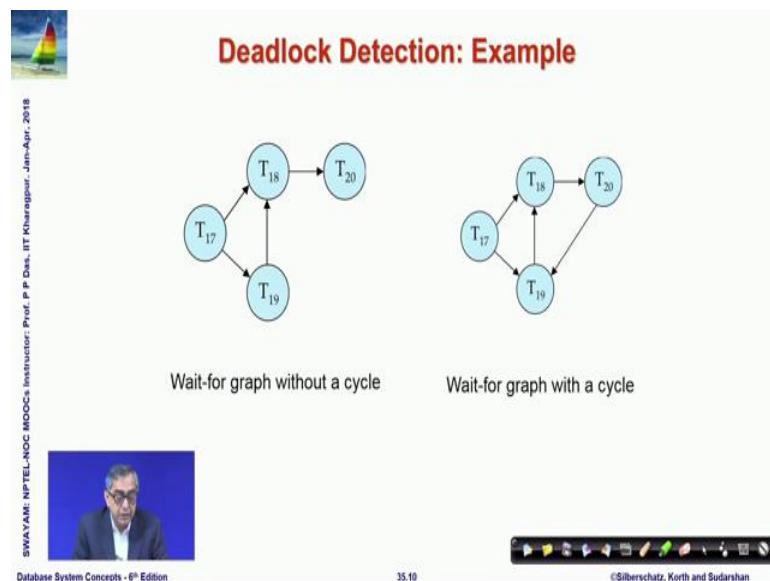
So, this is how this is kind of a dynamic graph the wait for graph is a kind of dynamic graph which will regularly keep getting updated. Now, naturally from the description of this graph, you can understand that a deadlock if a deadlock has to happen then this graph must have a cycle. So, if at any instant the graph has a cycle then there is a deadlock; otherwise the graph will grow and shrink grow and shrink it will keep on happening that way.

So, it is important to ensure that this graph remains acyclic which now this is dynamically happening hundreds of transactions, transactions are getting created, they are getting committed, aborted, they are requesting locks they are releasing locks and so on. So, how do you ensure that the graph at every stages is remaining a cyclic or a cycle has happened and therefore, a deadlock is actually happening.

So, what you will need to do is periodically run another process which invokes the deadlock-detection in the graph that is it looks for the cycles, and the cycle is there, then

you have to do some strategy to roll back about one of the transactions, and break the cycle and then so that the other transaction can proceed.

(Refer Slide Time: 11:29)



So, these are examples of the wait for graph. For example, here on the left as you see if you if I may point out in the left, if we see that T₁₇ is waiting for T₁₈ and T₁₉, T₁₈ is waiting for T₂₀. So, eventually and T₂₀ is waiting for none. So, at some point of time T₂₀ will be done and when that is done then T₁₈ would be able to proceed. And if T₁₈ is able to proceed then T₁₉ would be able to proceed, and then T₁₇ would be able to proceed.

So, there is no possibility of a deadlock, whereas in here if you look in the graph on right, then you can see that between these three they are waiting on each other. So, no matter how long you wait this will this deadlock will never be broken, and the deadlock-detection system has to detect this cycling and decide to abort one of these transactions and so that the rest of the transactions can progress. So, this is a simple deadlock-detection mechanism.

(Refer Slide Time: 12:44)

Deadlock Recovery

- When deadlock is detected :
 - Some transaction will have to be rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost
 - Rollback -- determine how far to roll back transaction
 - ↳ Total rollback: Abort the transaction and then restart it
 - ↳ More effective to roll back transaction only as far as necessary to break deadlock
 - Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation

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

Database System Concepts - 8th Edition

35.11

©Silberschatz, Korth and Sudarshan

So, when the deadlock is detected there has to be a recovery. So, trump transactions will have to be rolled back to break the deadlock. And so there is a there is a strategy required to select which transaction must rollback; naturally that should be done based on the minimum cost that is you do not want because if you roll back then the recomputation naturally because you have to restart and do that transaction again.

So, you have to in terms of rollback, you have to determine how far to rollback that transaction. There can be a total one, so that you roll back the whole transaction abort and then restart it or you can roll back to a previous point, we discussed notions of safe point in the transaction program.

And so it is be more effective to roll back transaction only as far as necessary to break the deadlock, you may not need to roll back everything. Maybe this is this transaction is participating in the deadlock, because it is holding some exclusive lock which it took three instructions before. But before that it has done 300 instructions it is not necessary to rollback the whole of the 300 instructions, you can just roll back up to the point where it took that exclusive lock which is creating the problem.

So, that those are some of the strategies which can improve the throughput and minimize the possibility of starvation.

Starvation will again happen if the same transaction is chosen as a victim to be rolled back every time, which the possibility exists. And so the number of roll backs is also usually kept as a cost factor. So, when you roll back a transaction, you also keep a number saying that how many times this transaction has been rolled back.

So, higher that cost becomes then you would like to avoid doing the rollback for that transaction because so that it does not wait infinitely in terms of (Refer Time: 14:45) starvation. So, these are some of the simple strategies that roll back the deadlock recovery can be done.

(Refer Slide Time: 15:04)

The slide has a title 'Timestamp-Based Protocols' in red at the top right. On the left is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp $TS(T_i)$, a new transaction T_j is assigned time-stamp $TS(T_j)$ such that $TS(T_j) < TS(T_i)$.
- The protocol manages concurrent execution such that the time-stamps determine the serializability order
- In order to assure such behavior, the protocol maintains for each data item two timestamp values:
 - **W-timestamp(Q)** is the largest time-stamp of any transaction that executed **write(Q)** successfully
 - **R-timestamp(Q)** is the largest time-stamp of any transaction that executed **read(Q)** successfully

At the bottom left is a small video thumbnail showing a person speaking. The footer contains the text 'SWAYAM: NPTEL-NOC's Instructional Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018', 'Database System Concepts - 8th Edition', '35.13', and '©Silberschatz, Korth and Sudarshan'.

So, having talked about the prevention detection and recovery from deadlocks let us quickly look at a simple time-based protocol in contrast to the two phase locking protocol we had earlier. This protocol does not lead to deadlock. So, what you do in here is each transaction is issued a timestamp when it enters the system. So, hold at the transaction, less is the value of the timestamp so that is a simple. So, time goes in the increasing order.

Now, the protocol manages a concurrent execution such that timestamps determine they themselves will determine the serializability order, they will determine in which order the transaction should occur. And for that for each data item two timestamp values are maintained; one is a right time-stamp on the data item queue, another is a read

timestamp. So, this is the latest read write and read times for the data item. So, w timestamp Q is the largest time stem of any transaction that executed a write Q successfully. So, naturally what it means the largest timestamp means the latest write that has happened. Similarly, it keeps it latest read.

(Refer Slide Time: 16:15)

Timestamp-Based Protocols

- The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order
- Suppose a transaction T_i issues a **read(Q)**
 1. If $TS(T_i) \leq W\text{-timestamp}(Q)$, then T_i needs to read a value of Q that was already overwritten.
 - Hence, the **read** operation is rejected, and T_i is rolled back
 2. If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the **read** operation is executed, and **R-timestamp(Q)** is set to $\max(R\text{-timestamp}(Q), TS(T_i))$

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

Database System Concepts - 6th Edition

35.14 ©Silberschatz, Korth and Sudarshan

Now, using that you build up this protocol, so it is again looks only at conflicting read and write operations, and they are executed in timestamp order. So, let us suppose that let us consider the case of read. So, a transaction T_i has issued a read.

Now, if that the timestamp of $T_i \leq W\text{-timestamp}(Q)$ which means that **W-timestamp(Q)** is the latest write. And $TS(T_i)$ is a timestamp of the transaction. So, the transaction is older than the latest write. So, the transaction T_i needs to read a value that was already overwritten because the latest write has happened after the transaction. So, this read operation can be rejected and T_i will be rolled back.

If it is in contrast, if the timestamp of the transaction is greater than the latest right time $W\text{-timestamp}(Q)$ then the read operation is executed and since we are doing a read operation. So, this becomes the latest read operation and therefore, the read timestamp **R-timestamp(Q)** is set to the maximum of the current read timestamp and the timestamp of the transaction. Mind you here we the timestamp one confusion that may come to your mind is are we looking at the exact time when the read has happened or when the write

has happened, no, we are all of this reasoning is happening with the timestamp of the transaction.

So, whenever it started. So, it is a older transaction and newer transition that we are reasoning with. So, when you update R-timestamp then you are the R-timestamp already has a value which is the timestamp of the latest transaction that has read that value and $TS(T_i)$ is the timestamp of the transaction that is read it now.

So, it is not always that since this read is the last read you will update this. So, by the sense of latest what I mean is the latest in the sense of the timestamp of the transaction that is reading it. So, you will compute that in terms of finding the maximum of the current read timestamp and the timestamp of this transaction.

(Refer Slide Time: 18:48)

The slide is titled "Timestamp-Based Protocols (Cont.)" in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list under a heading: "Suppose that transaction T_i issues **write(Q)**". The list includes three numbered points:

1. If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced
 - Hence, the **write** operation is rejected, and T_i is rolled back
2. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q
 - Hence, this **write** operation is rejected, and T_i is rolled back
3. Otherwise, the **write** operation is executed, and $W\text{-timestamp}(Q)$ is set to $TS(T_i)$

At the bottom of the slide, there is footer text: "SWAYAM: NPTEL-NOCO Instructor: Prof. B P Das, IIT Kharagpur - Jan-Apr. 2018", "Database System Concepts - 6th Edition", "35.15", and "©Silberschatz, Korth and Sudarshan". There is also a navigation bar with icons for back, forward, search, and other presentation controls.

Write is a little bit more complex, but we can reason in the same way. So, if T_i issues a write (Q). And if the timestamp of $T_i < R\text{-timestamp}(Q)$ that is if this transaction is it is less so it is older than the read timestamp that is it is older than the transaction that read Q last, the youngest transaction that read the value of Q .

So, then the value Q that T_i is producing was needed earlier, it is trying to write, but already a newer transaction has used the value. And the system assumed that what the T_i was supposed to write was not available was not produced, hence the write operation is rejected T_i does not should not write this and T_i will be rolled back.

Second case if the transaction T_i has a timestamp which is less than the write timestamp. So, which means that this transaction is older than the transaction that has done the last write; So, T_i is attempting to write an absolute value of Q , and hence this write operation is again rejected and T_i is rolled back. Otherwise, in other cases, the write operation is executed and the write timestamp will be set to the timestamp of this transaction which has written it. So, this is a very simple protocol for read and write.

(Refer Slide Time: 20:36)

Example Use of the Protocol

A partial schedule for several data items for transactions with timestamps 1, 2, 3, 4, 5

T_1	T_2	T_3	T_4	T_5
read (Y)	read (Y)			read (X)
		write (Y) write (Z)		read (Z)
read (X)	read (Z) abort		read (W)	
		write (W) abort		write (Y) write (Z)

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

And here is an example shown in terms of this protocol. For example, this wanted to do a read and this so this is the that this is a time where the transaction had started so and this was the write that. So, when this read is happening this is the so this is naturally this is at hold at transaction than T_3 . So, this at this point, the write timestamp is that of T_3 and this is an older one, so it was trying to read that, so this was aborted.

Similarly as you see here if I clean and start again if we look at write W this is trying to do read and T_4 , so read will this read has a timestamp which is of T_4 which is later than the timestamp of T_3 . So, this gets aborted. So, you will need to spend a little bit of time to convince yourself that this will never actually ensure never actually lead to any deadlock, and it is a very effective serializable and simple strategy to ensure serializability while it avoids the deadlock.

(Refer Slide Time: 22:08)

Correctness of Timestamp-Ordering Protocol

■ The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:

```
graph LR; A((transaction with smaller timestamp)) --> B((transaction with larger timestamp))
```

Thus, there will be no cycles in the precedence graph

- Timestamp protocol ensures freedom from deadlock as no transaction ever waits
- But the schedule may not be cascade-free, and may not even be recoverable

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

Database System Concepts - 8th Edition

35.17

©Silberschatz, Korth and Sudarshan

At all the timestamp ordering protocol itself guarantees the serializability, so the transaction with the smaller timestamp will lead to transaction with larger timestamp, because those are the more recent transaction. So, since the ordering is always in this manner, there cannot be any cycle in this precedence graph, because if they are cycle then naturally, somewhere you are you will be coming from newer to an older transaction which is not allowed in this protocol.

So, there cannot be a cycle and so this ensures that there cannot be deadlock in this time-based protocol, but the schedules that they produce they may not be cascade free. And actually examples can be shown that they may not even be recoverable there may be some irrecoverable schedules that get produced through this time-based protocol.

(Refer Slide Time: 23:10)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains two bullet points:

- Explained how to detect, prevent and recover from deadlock
- Introduced a time-based protocol that avoids deadlocks

On the far left, vertical text reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 8th Edition", "35.18", and "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is at the bottom.

So, to summarize we have tried to take a look into explaining what are the different ways to prevent deadlock; Some of the strategies and specifically we focused on the time-based strategy. So, there are some strategies which are based on the order of accesses the data items ordering of data items and so on. And we have specifically focused on time-based strategies.

And from that there are multiple time-based strategies which can prevent deadlock. And in case the deadlock has happened then we have discussed a simple wait for graph data structure and algorithm to be able to detect that the deadlock has happened. And if once this has been detected, we have talked about basic strategy to recover from that that is how do you decide what are the what is a good candidate victim transaction which should be rolled back, which should be aborted.

And on that study we have presented a simple time-based protocol which maintains the timestamp of transactions to decide the ordering in terms of read and write and deciding as to whether you should continue doing a read or write or you should abort the read and write attempt; And thereby ensuring that deadlocks do not happen in the system though there may be other problems in this in terms of having cascading rollback or having some irrecoverable schedules at times.

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

Lecture – 36
Recovery/1

Welcome, to module – 36 of Database Management Systems. In this module and the next, we will talk about recovery in databases.

(Refer Slide Time: 00:27)

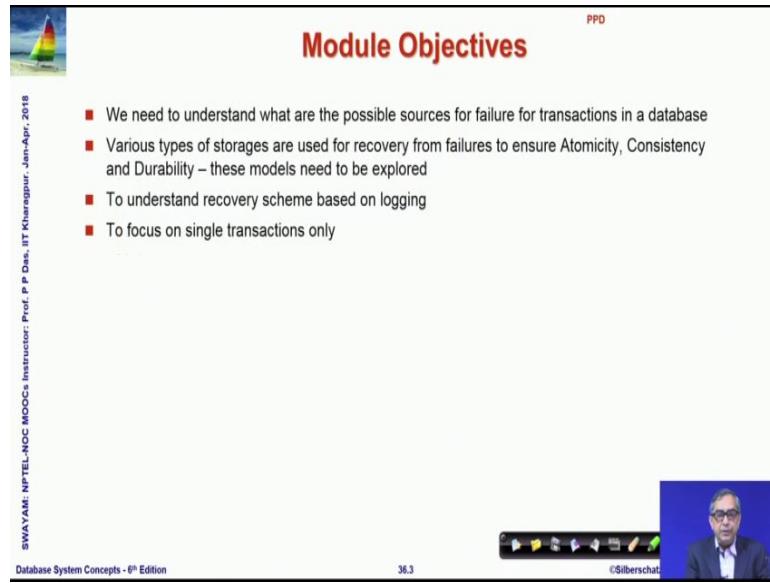
The slide is titled "Week 07 Recap" in red at the top right. It features a small sailboat icon in the top left corner. The main content is organized into two columns of bullet points:

<ul style="list-style-type: none">▪ Module 31: Transactions/1<ul style="list-style-type: none">○ Transaction Concept○ Transaction State○ Concurrent Executions▪ Module 32: Transactions/2: Serializability<ul style="list-style-type: none">○ Serializability○ Conflict Serializability▪ Module 33: Transactions/3: Recoverability<ul style="list-style-type: none">○ Recoverability and Isolation○ Transaction Definition in SQL○ View Serializability○ Complex Notions of Serializability	<ul style="list-style-type: none">▪ Module 34: Concurrency Control/1<ul style="list-style-type: none">○ Lock-Based Protocols○ Implementing Locking▪ Module 35: Concurrency Control/2<ul style="list-style-type: none">○ Deadlock Handling○ Timestamp-Based Protocols
--	---

At the bottom left, there is vertical text: "SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number "36.2" is also visible.

In the last week, we have talked at length in terms of the transactions and the concurrency control. And we will see how the acid properties of the transaction can be fulfilled using the different recovery schemes.

(Refer Slide Time: 00:41)



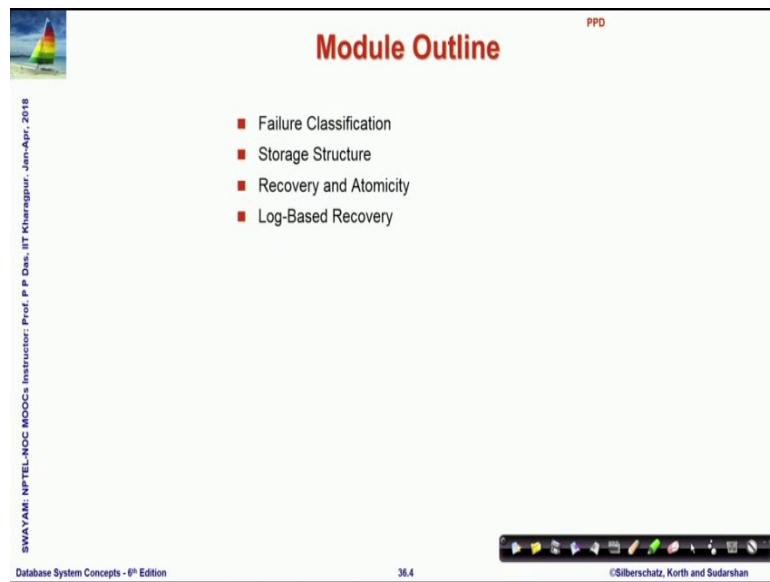
The slide features a sailboat icon in the top left corner and the text "PPD" in the top right corner. The title "Module Objectives" is centered in red. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". The main content area lists four objectives:

- We need to understand what are the possible sources for failure for transactions in a database
- Various types of storages are used for recovery from failures to ensure Atomicity, Consistency and Durability – these models need to be explored
- To understand recovery scheme based on logging
- To focus on single transactions only

At the bottom right is a video frame showing a man, identified as Prof. Silberschatz, and a navigation bar with icons.

To, specifically we will try to understand the different sources of failure and how the recovery can be facilitated by different storage structures particularly those different models of volatile and nonvolatile storages and we will take a look into recovery schemes that are based on logging mechanism and for this module we will focus only on single transactions.

(Refer Slide Time: 01:11)



The slide features a sailboat icon in the top left corner and the text "PPD" in the top right corner. The title "Module Outline" is centered in red. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". The main content area lists five topics:

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

At the bottom right is a navigation bar with icons.

So, these are the topics that will cover.

(Refer Slide Time: 01:17)

Database System Recovery

- All database reads/writes are within a transaction
- Transactions have the "ACID" properties
 - Atomicity - all or nothing
 - Consistency - preserves database integrity
 - Isolation - execute as if they were run alone
 - Durability - results are not lost by a failure
- Concurrency control guarantees I, contributes to C
- Application program guarantees C
- Recovery subsystem guarantees A & D, contributes to C

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

Database System Concepts - 8th Edition

PPD

36.6 ©Silberschatz, Korth and Sudarshan

So, what we have looked at is all database writes and reads are within a transaction and transactions must satisfy the ACID properties and in terms of the concurrency control we have already seen that concurrency in a controlled guarantees isolation of transactions and in a certain way it contributes to achieving maintaining consistency. Application programs are heavily responsible for guaranteeing consistency, but to really guarantee the atomicity and durability of the data that the transactions reads and write the recovery sub system is required and it also contributes to the consistency property.

(Refer Slide Time: 01:56)

Failure Classification

- **Transaction failure:**
 - **Logical errors:** transaction cannot complete due to some internal error condition
 - **System errors:** the database system must terminate an active transaction due to an error condition (for example, deadlock)
- **System crash:** a power failure or other hardware or software failure causes the system to crash
 - **Fail-stop assumption:** non-volatile storage contents are assumed to not be corrupted as result of a system crash
 - ▶ Database systems have numerous integrity checks to prevent corruption of disk data
- **Disk failure:** a head crash or similar disk failure destroys all or part of disk storage
 - Destruction is assumed to be detectable
 - ▶ Disk drives use checksums to detect failures

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

Database System Concepts - 8th Edition

36.7 ©Silberschatz, Korth and Sudarshan

So, let us look at if we are talking about recovery and the phase of failure. So, let us look at what are the generic types of failures that can happen one is the type transactions can fail. A transaction can fail due to logical error due to some internal error or it might fail due to some system error. So, that the system must terminate the transaction, we have talked about several situations where deadlock might happen and the transaction needs to be rolled back that is a kind of transaction failure error.

The second possible error can happen if there is a crash in the system. A system can crash due to hardware failure power failure software failure. So, we try to make fail stop assumptions that nonvolatile contents are assumed to be eh corrupted and database systems consequently have to involve a number of integrity checks to prevent the corruption of data.

And, the third broad category of failures happen with disk failure a disk might itself fail it is hardware may fail the head may crash and when that happens then the destruction is assumed to be detectable we must be able to detect such failures. There are checksums and other mechanisms for detecting failures, but broadly these are the three types of failures that a database system can go through.

(Refer Slide Time: 03:23)

The slide has a title 'Recovery Algorithms' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list of points:

- Consider transaction T_i that transfers \$50 from account A to account B
 - Two updates: subtract 50 from A and add 50 to B
- Transaction T_i requires updates to A and B to be output to the database
 - A failure may occur after one of these modifications have been made but before both of them are made
 - Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
 - Not modifying the database may result in lost updates if failure occurs just after transaction commits
- Recovery algorithms have two parts
 1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
 2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

At the bottom left, there is vertical text: 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr- 2018'. At the bottom right, there are navigation icons and the text 'Database System Concepts - 8th Edition', '36.8', and '©Silberschatz, Korth and Sudarshan'.

So, in view of that ifs one or more of this failures happen then we need mechanisms to recover from that let us consider a very simple situation of a transaction which we saw earlier to that a transaction T_i transfers dollar 50 from account A to account B and

therefore, two updates have to happen; A has to get debited and B has to get credited. So, the transaction T_i requires updates to A and B that are happening that must be written that must be output to the database in a permanent manner. So, if failure may occur after one of these modifications have happened and before both of them are made, so that is one possibility. One possibility is we can get we have modified the database without ensuring that the transaction will necessarily commit, but the database has been checked transaction may not have committed. So, that will leave the database inconsistent because the transaction will have to be rolled back or it may so happen that database has not been modified and the, but the transaction has committed.

So, if the failure occurs at that point then there will be some lost updates. So, the recovery algorithms strategy has to primarily take care of two things; one is during the normal transactions it has to collect enough information so that the recovery from failures can be done. So, one is what we need to do while in normal transaction is going on because during that time we need to have enough data so that we can recovery in the phase of failure and the second set of actions are actions that can once a failure has happened to recover the database so that we can go back to a consistent state and ensure the atomicity consistency and durability of the transaction.

So, before we get into these discussions of the different recovery algorithms let us quickly look into this storage model that we are assuming.

(Refer Slide Time: 05:30)

The slide has a header 'Storage Structure' in red. On the left, there is a small image of a sailboat on water. The right side contains three bulleted sections: 'Volatile storage', 'Nonvolatile storage', and 'Stable storage', each with associated bullet points. At the bottom, there is a video player showing a man speaking, and the footer includes course and copyright information.

Storage Structure

■ **Volatile storage:**

- does not survive system crashes
- examples: main memory, cache memory

■ **Nonvolatile storage:**

- survives system crashes
- examples: disk, tape, flash memory, non-volatile (battery backed up) RAM
- but may still fail, losing data

■ **Stable storage:**

- a mythical form of storage that survives all failures
- approximated by maintaining multiple copies on distinct nonvolatile media

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

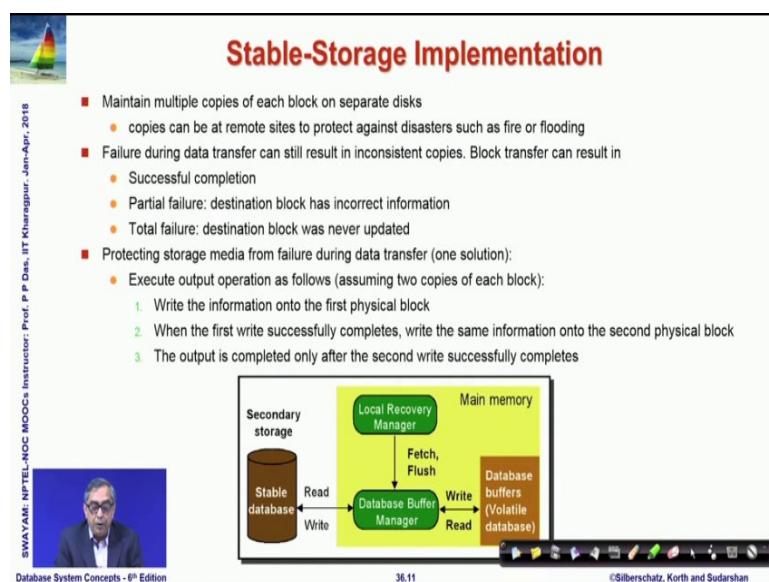
Database System Concepts - 8th Edition

36.10 ©Silberschatz, Korth and Sudarshan

We know there is volatile storage which we have discussed about that we know this nonvolatile storage which disk, tape, flash and all that volatile storage disappears whenever system crashes and non-volatile storage is supposed to survive the system crash, but it may still fail it may still cause loss of data.

So, we also consider a third kind of storage which is notionally known as stable storage. It is called a mythical form of storage where we assume that it will survive all kinds of failures. Now, naturally this in ideality this can never happen, but we can approximate this by maintaining multiple copies of the same data on distinct non-volatile media and the stable storage would be assumed to be one available component in the database system for making the recovery systems work.

(Refer Slide Time: 06:21)



So, as you can see in this diagram below. So, we are trying to explain more of what is the stable storage. So, this is on the secondary storage so, you have a stable database. So, kind of approximates that it will never fail whereas, on a routine basis things happen in terms of database buffers which are basically volatile databases, the volatile memory. So, now the fig so, what we do is we maintain multiple copies of each block of data and keep them on separate disk. So, even if one disk fail that is possible to recover from other disk. There are different kinds of the multiplicity that can be done even it can be located at a remote location so that even if there is a fire or flooding the database can be

recovered, but in principle will assume that the multiple copies are not all copies can fail at the same time.

So, it can now this will ensure the data has already been written then it is guarantee that it will stay we have talked about rate systems, but what happens if the failure happens during the data transfer where the result is still in transient state in it will live with transient copies. So, block transfer in general can result in either in successful completion or in partial failure, where the destination block actually has in current information or total failure where destination block could not be updated at all. So, to protect against the media again such failures during the data transfer the one possible solution could be and we assume that there are only two copies of each block it could you could have multiple copies to give you more resiliency against failure.

So, if we have two copies and the strategy could go like this that write the information onto the first physical block then once that is successfully completed then you write the same information on the second or physical block and the output is completed only after the second write the physical block is successfully completed. So, that is what we need to guarantee.

(Refer Slide Time: 08:45)

The slide has a title 'Stable-Storage Implementation (Cont.)' in red. Below the title is a sub-section header 'Protecting storage media from failure during data transfer (cont.):'. A bulleted list follows:

- Copies of a block may differ due to failure during output operation. To recover from failure:
 1. First find inconsistent blocks:
 - 1. *Expensive solution:* Compare the two copies of every disk block
 - 2. *Better solution:*
 - Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk)
 - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these
 - Used in hardware RAID systems
 2. If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy
 3. If both have no error, but are different, overwrite the second block by the first block

At the bottom left is a small video thumbnail showing a person speaking. The bottom right contains navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

Now, to protect against that happens if during this transfer with during this write if some output operation is some failure happens. So, to recover from that you need to find out what are the blocks which are inconsistent, because we have kept two or more

copies so you have to compare two copies of every disk block have kept them at separate disks and see which one whether there has been some inconsistency. Now, this is theoretically ok, but this is very expensive because there are so many different blocks.

So, what is typically done a better solution is while you are actually doing the disk write where you are actually doing the output on a in the process of doing the output then you record these writes on a nonvolatile storage say non-volatile ram or special area of the disk and use this information during the recovery to find the blocks that are inconsistent and only compare those copies. So, that will be naturally much faster because memory as you know is much faster to access than the disk and these are strategy which is typically used in the rate system we have discussed earlier.

So, if either of either copy of a inconsistent block is detected to have some kind of an error, to checksums to the error then over write by the other copy, but if both have no error, but are different then overwrite the second one by the first one. So, this will make sure that you always have even if there is a transient failures you can take care of that you know what is wrong and you can take care of and correct that.

(Refer Slide Time: 10:25)

The slide has a title 'Data Access' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list of points about data access:

- **Physical blocks** are those blocks residing on the disk
- **System buffer blocks** are the blocks residing temporarily in main memory
- Block movements between disk and main memory are initiated through the following two operations:
 - **input(B)** transfers the physical block B to main memory
 - **output(B)** transfers the buffer block B to the disk, and replaces the appropriate physical block there
- We assume, for simplicity, that each data item fits in, and is stored inside, a single block

At the bottom left is a video frame showing a man speaking, with the text 'SWAYAM-NPTEL-NCCOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center is the text 'Database System Concepts - 6th Edition' and '36.13'. At the bottom right is the text '©Silberschatz, Korth and Sudarshan' and a set of navigation icons.

Now, to make this kind of a mechanism work we resort to a very simple module of data access. We assume that there are physical blocks on the disk that are on the non-volatile permanent storage and that is where finally, you want your data to decide,

but you also assume that there are system buffer blocks; the blocks that we decide temporarily in the main memory, so, they can be used in the in transit.

So, when you move the block between the disk and the main memory that is initiated by an input operation. So, you are doing an input so, all the physical block a that is disk a block physical block B is brought into the main memory or you have a output operation which transfers first a buffer block B to the disk and replaces the appropriate physical block there. So, these operations when you move physical blocks with the disk you call them as input and output. So, and we are making some assumption that the data that we want to write is small enough so that it fits into a block otherwise there are several schemes of or you know spread your data over multiple blocks.

(Refer Slide Time: 11:40)

The slide has a header 'Data Access (Cont.)' in red. On the left is a small logo of a sailboat. The content is organized into sections with bullet points:

- Each transaction T_i has its private work-area in which local copies of all data items accessed and updated by it are kept
 - T_i 's local copy of a data item X is denoted by x_i
 - B_X denotes block containing X
- Transferring data items between system buffer blocks and its private work-area done by:
 - **read(X)** assigns the value of data item X to the local variable x_i
 - **write(X)** assigns the value of local variable x_i to data item $\{X\}$ in the buffer block
- Transactions
 - Must perform **read(X)** before accessing X for the first time (subsequent reads can be from local copy)
 - The **write(X)** can be executed at any time before the transaction commits
- Note that **output(B_x)** need not immediately follow **write(X)**. System can perform the **output** operation when it deems fit

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOCO's Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr-2018'. At the bottom center: 'Database System Concepts - 8th Edition' and '36.14'. At the bottom right: '©Silberschatz, Korth and Sudarshan'.

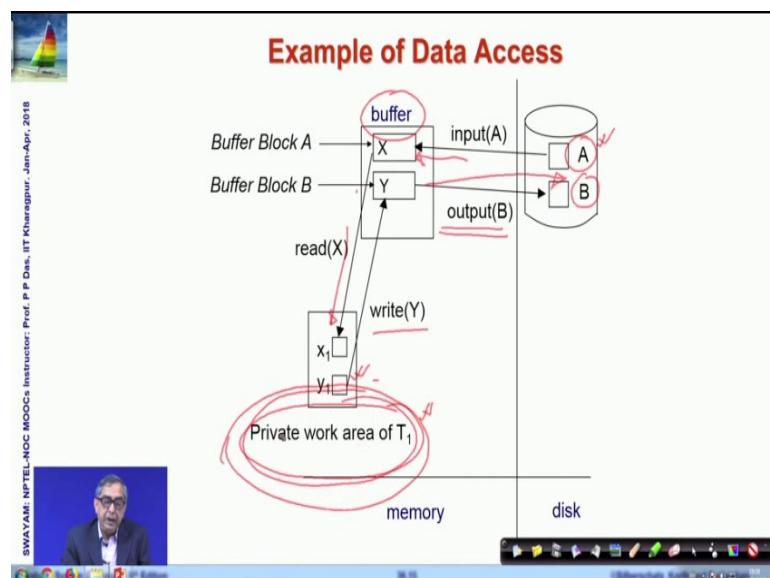
Now, the other part is each transaction on the other side is assumed to have a private work area. So, in the private work area that transaction actually gives local copies and these local copies say you have a data item X, so, you say that for transaction T_i the copy of that data item X is x_i and say, B_x is the block that contains X. So, B_x is the physical block and then you can transfer data between the transactions private area and this buffer block in terms of read and write operations.

So, we have two kinds of operation; one is input output which is between the memory and the physical block that is the disk and the other is read write operation which is between the transactions private area and the system buffered blocks. So, the transaction

must perform read before accessing X for the first time and once it is done that it has a local copy now and therefore, subsequent reads can happen from the local copy and the write can be executed at any time before the transaction actually commits.

So, let us look at also it is a fact is that the when I want to actually output the block that contains X, I mean your item X to be finally, written to disk the output B x need not immediately happen after you write. So, you are doing it in two stages, from the transactions private area to the system buffer, to the system buffer to the disk. So, first is write the next is output, but this may not actually follow immediately once the data exists in the system buffer it may be actually output on a at a later time whenever it is then fit to do that.

(Refer Slide Time: 13:32)



So, let us take a very quick looks schematically here to make things some simple understandable. So, they are data items A and B on the disk that we are talking of. So, if I do an input operation then I am actually trans and this is the buffer this is the system buffer. So, this is kind of a common buffer where you can keep data, we will see how to manage this system buffer and this is the private work area of a transaction say T₁.

So, to process of read I mean if T₁ wants to if T₁ wants to read A then the that read initiation will bring A onto the buffer area as X and then it will read X as x₁ in its private area, it will this is where it will do the work. This is a private area where T₁ will do the work and possibly it has generated a write item y with which needs to go back to the

disk. So, again we will do a write to the buffer area and then at a later point there will be an output which will take this Y back to the disk.

So, this will ensure that the transaction can after reading the transaction can independently do the write to the buffers and outputs can happen independently of that either before that transaction commits or even after the transaction commits there are difference in situation there are different protocols that are followed and we will see through, but this is the basic simple model that will regularly be used.

So, please keep in mind we will talk about often will talk about three areas work area the private work area of a transaction this is an memory and the system buffer blocks where the data is temporarily deciding on the way of being read or on the way of being written and the system disk where the physical block exists. And, this is the path way through this system buffer that the read writes will output will happen and please remember that we will use the term read write when it is between the private work area of a transaction and the system buffer block and will talk about input output when it is between in terms of the physical block with the disk.

So, the data access you can I have already explained. So, in terms of the data access these are the steps that the transaction will do to read or write as I have already explained. Now, in terms of now, let us see that how will in the background of such a storage access how will the recovery happen and how will the atomicity be guaranteed.

(Refer Slide Time: 16:20)

The slide has a title 'Recovery and Atomicity' in red at the top right. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists several bullet points:

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself
- We study **log-based recovery mechanisms** in detail
 - We first present key concepts
 - And then present the actual recovery algorithm
- Less used alternative: **shadow-paging**
- In this Module we assume serial execution of transactions
- In the next Module, we consider the case of concurrent transaction execution

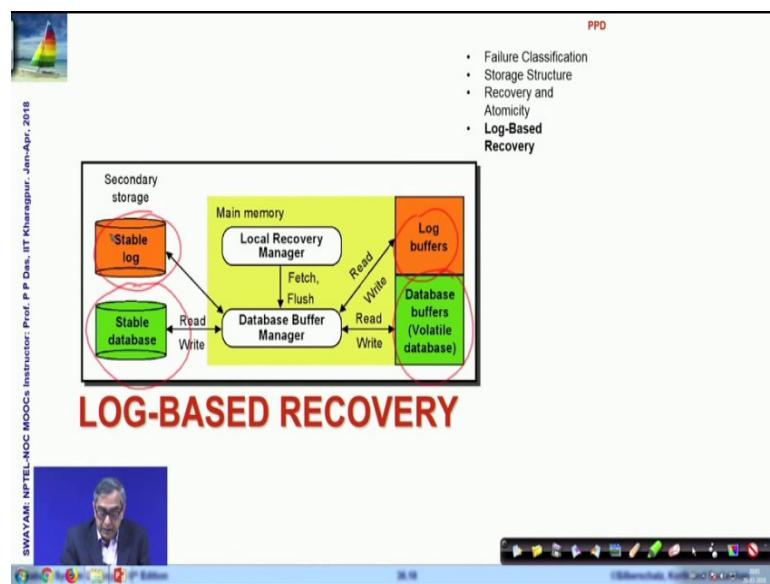
At the bottom left is a video player showing a person speaking, with the text 'Database System Concepts - 8th Edition'. At the bottom right is a navigation bar with icons for back, forward, search, and other presentation controls. The footer also includes the text '©Silberschatz, Korth and Sudarshan'.

So, to ensure atomicity in the phase of failure we need to output information describing the modifications to stable storage with input modifying database itself. So, what we are saying that to be able to recover that we should write the changes to the stable storage you recall that stable storage something that is assumed to be not failing without actually modifying database.

Now, we do a very simple mechanism which is called a log based recovery mechanism. So, we will first talk about this log based recovery mechanism what are the key concepts of logging and redo undo redo kind of operations and present the actual recovery algorithm. There are other alternatives also like shadow paging we will not discuss about that and I would like to again remind you that in this module we are talking about single transactions at a time, serial execution.

In the next module we will talk about concurrency of the; I mean the behavior of recovery algorithms and in the case of concurrent transactions.

(Refer Slide Time: 17:29)



So, now let us talk about the log based recovery mechanism. So, in the log based recovery mechanism you can see this is the basic this is your stable database which you want to make use of, these are your buffers you talked off and we will have certain logs the information of what have been doing in terms of the log buffers and the also is stable log which is a log that is written in the stable database. So, once we understand what is logging you will understand this, but I just wanted to show you that

like the data there are buffer copies as well as stable database copies in terms of log also there will be buffer copies as well as stable, log copies.

(Refer Slide Time: 18:18)

The slide has a title 'Log-Based Recovery' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points about log-based recovery:

- A **log** is kept on stable storage
 - The log is a sequence of **log records**, which maintains information about update activities on the database
- When transaction T_i starts, it registers itself by writing a record $\langle T_i, \text{start} \rangle$ to the log
- Before T_i executes **write(X)**, a log record $\langle T_i, X, V_1, V_2 \rangle$ is written, where V_1 is the value of X before the write (the **old value**), and V_2 is the value to be written to X (the **new value**)
- When T_i finishes its last statement, the log record $\langle T_i, \text{commit} \rangle$ is written
- Two approaches using logs
 - Immediate database modification
 - Deferred database modification

At the bottom left, there is a small video thumbnail showing a man speaking, with the text 'SWAYAM-NIITL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018'. At the bottom center, it says 'Database System Concepts - 8th Edition'. At the bottom right, it shows a Mac OS X dock with various icons and the text '36.19 ©Silberschatz, Korth and Sudarshan'.

So, a log is kept in the stable storage, it is a sequence of records. So, log is basically a record of what and. So, it is like if am doing some task we have always every task I do I keep a record of what am actually be doing and that is called the logging. So, when a transaction starts I write a log record which puts the transaction ID say T_i and then puts a keyword start to the log. So, that indicates that the transaction T_i has started and when it is about to execute a write, so, ma before it has actually executed the write, then I write a log record which looks like this which is.

So, here if we look carefully this is the idea of the transaction X is the data item that you want in to write V_1 is the current value of the data item which kind of we can say is the old value and V_2 is the value that we want to actually write. So, here you can see that we are clearly keeping a track of what we are writing and in that process what is the original value that would get changed. So, that is the main important factor of this logging that every with every write you remember as to what value was originally there and what value we have actually changed it to in that transaction.

Now, in this process finally, when that the transaction finishes the last statement of the log record is T_i commit. So, that actually is a meaning of committing a transaction when

this log record is written out. So, that is. So, a log will have start then different write log records and then finally, a commit log record.

So, there are basically two approaches of using log, one is called immediate database modification this is what we would follow here and there is a differed database modification.

(Refer Slide Time: 20:19)

Database Modification

- The **immediate-modification** scheme allows updates of an uncommitted transaction to be made to the buffer, or the disk itself, before the transaction commits
- Update log record must be written **before** a database item is written
 - We assume that the log record is output directly to stable storage
- Output of updated blocks to disk storage can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written
- The **deferred-modification** scheme performs updates to buffer/disk only at the time of transaction commit
 - Simplifies some aspects of recovery
 - But has overhead of storing local copy
- We cover here only the immediate-modification scheme

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jun-Apr, 2018

Database System Concepts - 8th Edition

36.20

©Silberschatz, Korth and Sudarshan

In the immediate modification scheme the ma it allows updates of an uncommitted transaction to be made to the buffer or the disk itself before the transaction commit. So, before the transaction has committed that is before the $\langle T_i, \text{commit} \rangle$ log record has been written at that point itself you allow the updates of the transaction to be made to the buffer or the disk and the update log record must be written before the database is actually written. So, you must first write the log and then actual database item. So, and we assume that the log record is output directly to the stable storage. So, that it is not there is no possibility of is getting lost.

Now, output of the updated blocks to disk storage can take place, that is the final actual output this is where we have written the log that that am doing this change, but the actual change we can take place any time before the transaction commits or even after the transaction commits. If you follow this ma protocol then you say you are in the immediate modification scheme and in fact, the order in which the blocks are output that finally, written to the disk may be different from the order in which they were originally

written, but the log records the will have to be written before these each one of this output are done.

In the deferred modification scheme the change updates are performed to buffer and disk only at the time of transaction commit not any time before that. So, that simplify some aspects of recovery, but it has other issues. So, we will not talk about this scheme, just know that there is an alternate scheme for doing things.

(Refer Slide Time: 22:03)

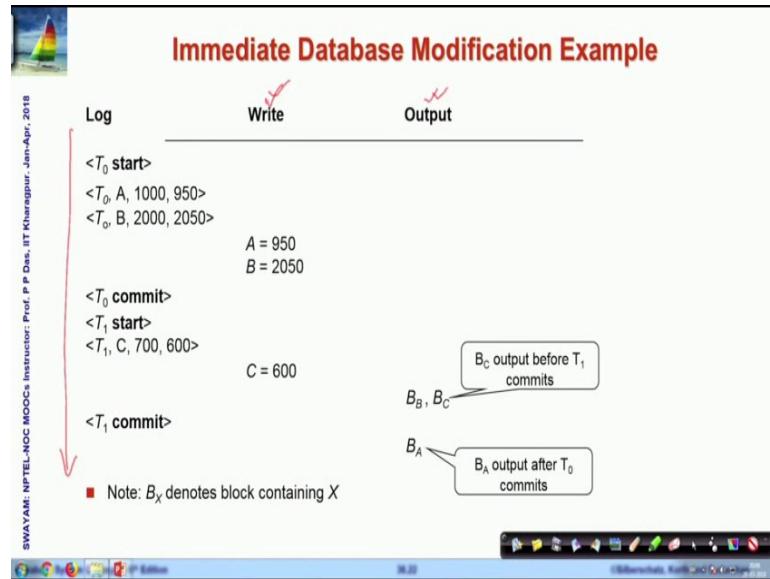
The slide has a title 'Transaction Commit' in red at the top right. To the left of the title is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018'. At the bottom left is a video frame showing a man with glasses speaking. The bottom of the slide contains navigation icons and the text 'Database System Concepts - 8th Edition' on the left, '36.21' in the center, and '©Silberschatz, Korth and Sudarshan' on the right.

- A transaction is said to have committed when its commit log record is output to stable storage
 - All previous log records of the transaction must have been output already
- Writes performed by a transaction may still be in the buffer when the transaction commits, and may be output later

So, now formally speaking what is transaction commit? A transaction commit is said transaction is said to have committed if its commit log record is output to the stable storage. That is T_i commit has gone to the stable storage is the meaning of the transaction has been committed. Obviously, all previous log records of the transactions must have been outputted already because those commit those outputs will have happen in the same order in which the actions are taken.

Now, the writes performed by the transaction may still be in the buffer. So, you have transaction is committed everything is done, but your actual writes that are performed may not have been outputted. They are they may still be in the buffer when the transaction commits and those may be output at a later point of time.

(Refer Slide Time: 22:52)



So, let us take an example here. Let us look at an example. So, here you see the log records and here is the sequence of write and output A_1 is happening. So, in the log record the transaction starts here. So, you have a log record of start, what is the meaning of this? The meaning of this is transaction T_0 is trying to write A and the current value is 1000 and it wants to change it to 950. So, this log record is written and you can see that the actual write actual write has not happened here, actual write is not done, but it is already has must like in the immediate database modification scheme it must write the log record before actually writing the output, actually doing the output or doing the write. So, this has happened here.

Similarly, the next one is another update transaction for B and actual writes have happened. So, which means the data has been written from the transactions private work area to the system buffer and then the transaction has transaction T_0 has done commits. So, at this point if you go up to this then the commit of the transaction is already completed and another transaction T_1 starts you please remember that we have said that we will we are using serial ma schedules only. So, only now another transaction can commit that has started and that has written log record for updating C from 700 to 600. So, there is a write for 600 then T_1 has commit.

In the meanwhile and at this stage, in the meanwhile these blocks have been output. So, they have actually been written the disk and you can understand that this block B B is a

block that contains the data of data item the updated value of data item B and ma this B C has the updated value of data item C. So, you can have see that actually these output of B is happening after the transactions is T_0 has committed whereas, for update of data you can see the output is happening af before the T_1 has committed. So, here it is happening after the commit, but here it is happening before the commit.

So, both of these are permitted both of these are allowed in terms of the protocol that we are following. And, you can also see that in terms of the order in which they were written A was written earlier, but A is output at a later point of time because that is a different sequence in which the system might decide for writing the buffer onto the disk.

So, this is the immediate database modification scheme through which we can write the logs.

(Refer Slide Time: 25:51)

Undo and Redo Operations

- Undo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the old value V_1 to X
- Redo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the new value V_2 to X
- Undo and Redo of Transactions
 - $\text{undo}(T_i)$ restores the value of all data items updated by T_i to their old values, going backwards from the last log record for T_i
 - Each time a data item X is restored to its old value V a special log record (called **redo-only**) $\langle T_i, X, V \rangle$ is written out
 - When undo of a transaction is complete, a log record $\langle T_i, \text{abort} \rangle$ is written out (to indicate that the undo was completed)

Handwritten notes on the right side of the slide:

- A vertical double-headed arrow between two lines of code.
- Handwritten values for X :
 - $X=10$ (with a circled '0')
 - $X=14$ (with circled '4' and '1')
 - $X=13$ (with circled '3')

Now, the question is we have written the logs. So, what is the use of those logs? Naturally, the use of those logs are in terms of two operations to which we say are undo operation and redo operation and undo operation is one which basically undoes the operation the effect of an update. So, while ma you have done you have if this is a log record then undoing, so, this meant that X was changed it had a value V_1 and it was changed to V_2 . If you undo that then the old value comes back to this old value comes back to X . So, that is if I undo this particular action the which was put in the log record then X will get back it is original value and redo is doing the same thing over again if I

redo for this log record then the value of V_2 will again reset on X. So, these are the two simple undo and redo operations which will help us achieve the recovery systems input.

So, what is meant by undo redo of transactions let us understand. So, when I undo a transaction T_i that restores the values of all data items updated by T_i to their old values. So, the values have been updated in this forward order. So, when you go to undo you will actually have to do that in the reverse order, because it is quite possible that X got 1 here then at some point it was updated to 17 then at some later point it was updated to 13. So, this update possibly had happened from 0, this update had happened from 1, this update had happened from 3. So, all those transaction records are there then you going backwards. So, you will first restore X back to 17 because this is then this back restoring back to 1 then going back to 0, in this order it will go on.

And, every time you restore you write that you write that out as a record which is known as redo, redo only record. So, you can see that here you are not trying to remember the original value you are just writing the value that you have written out in terms of the undo operation that is the old value and the going in this manner undo operation will terminate when you have come across the beginning of this one process, when it is complete then a log record $\langle T_i, \text{abort} \rangle$ is written out which says that the undo is actually over. So, this is the undo operation.

(Refer Slide Time: 28:36)

The slide has a title 'Undo and Redo Operations' in red. To the left is a small logo of a sailboat on water. On the right is a video frame showing a person speaking. The slide content is as follows:

■ Undo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the **old** value V_1 to X
■ Redo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the **new** value V_2 to X
■ Undo and Redo of Transactions

- $\text{undo}(T_i)$ restores the value of all data items updated by T_i to their old values, going backwards from the last log record for T_i
 - Each time a data item X is restored to its old value V a special log record (called **redo-only**) $\langle T_i, X, V \rangle$ is written out
 - When undo of a transaction is complete, a log record $\langle T_i, \text{abort} \rangle$ is written out (to indicate that the undo was completed)
- $\text{redo}(T_i)$ sets the value of all data items updated by T_i to the new values, going forward from the first log record for T_i
 - No logging is done in this case

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

For redo you said that the redo is doing the transactions doing the same instructions of the transactions in the same manner, it was done earlier. So, that unlike undo which goes backwards redo goes forward and it starts from the first log record of this transaction and goes on till the end and for this there is no separate logging for this operation.

(Refer Slide Time: 29:04)

The slide has a title 'Undo and Redo Operations (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points about undo and redo operations. At the bottom left, there is a video player showing a person speaking, with the text 'SWAYAM-NIPTEL-MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018'. At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls, along with the text 'Database System Concepts - 8th Edition' and 'Silberschatz, Korth and Sudarshan'.

- The **undo** and **redo** operations are used in several different circumstances:
 - The **undo** is used for transaction rollback during normal operation
 - ▶ in case a transaction cannot complete its execution due to some logical error
 - The **undo** and **redo** operations are used during recovery from failure
- We need to deal with the case where during recovery from failure another failure occurs prior to the system having fully recovered

Now, how will the undo redo operations be used? There are two major situations in which they are used one is undo is used transactions roll back have to roll back during normal operation. That is nothing has I mean there is no system failure or there is no data disk failure anything, but if the transaction has a normal failure that it cannot complete it is execution due to some logical error or because it has to roll back because of deadlock or something, then you what you do you just undo the whole effect of the transaction go backwards and keep on undoing. But, when the there is a failure there is a failure and you have to recover from that then undo and redo operations both will be required as we will soon see.

So, we also need to deal with the case where the recovery from failure while you are recovering from failure another failure happens. So, what do you do in that case that is more complicated will talk about that later?

(Refer Slide Time: 30:04)

The slide has a title 'Transaction rollback (during normal operation)' at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of steps:

- Let T_i be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of T_i , of the form $\langle T_i, X_j, V_1, V_2 \rangle$
 - Perform the undo by writing V_1 to X_j
 - Write a log record $\langle T_i, X_j, V_1 \rangle$
 - ▶ such log records are called **compensation log records**
- Once the record $\langle T_i, \text{start} \rangle$ is found stop the scan and write the log record $\langle T_i, \text{abort} \rangle$

At the bottom left, there is a photo of a man, and the text 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center, it says 'Database System Concepts - 8th Edition' and '36.25'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, first let us discuss what happens when you roll back a transaction during normal operation. So, let T_i be the transaction. So, you have to naturally do the undo because you have to undo the effect that it has already created. So, you will scan the log records from the end and for each log record which is kind of an update like this you will perform an update to restore the original value the old value and write out a redo only log record or which is called compensation log record which says that this has been undone to the value V_1 which is the original value of the transaction original value of the data item sorry.

Now, in going in this process backwards at some point of time you will reach come across $\langle T_i, \text{start} \rangle$ log record when you face come across that you write log record $\langle T_i, \text{abort} \rangle$ indicating that the undo of that transaction is over. So, this is the basic process of undoing the transactions during rollback.

(Refer Slide Time: 31:07)

Undo and Redo on Recovering from Failure

When recovering after failure:

- Transaction T_i needs to be undone if the log
 - contains the record $\langle T_i, \text{start} \rangle$,
 - but does not contain either the record $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$
- Transaction T_i needs to be redone if the log
 - contains the records $\langle T_i, \text{start} \rangle$
 - and contains the record $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$
- It may seem strange to redo transaction T_i if the record $\langle T_i, \text{abort} \rangle$ record is in the log
 - To see why this works, note that if $\langle T_i, \text{abort} \rangle$ is in the log, so are the redo-only records written by the undo operation. Thus, the end result will be to undo T_i 's modifications in this case. This slight redundancy simplifies the recovery algorithm and enables faster overall recovery time
 - such a redo redoes all the original actions including the steps that restored old value – known as **repeating history**

SWAYAM-NIPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
36.26 ©Silberschatz, Korth and Sudarshan

In the other case if you are recovering from a failure if there has been a failure then you do something which needs to be understood carefully. So, the transaction T_i needs to be undone if the log contains the record start $\langle T_i, \text{start} \rangle$, but it does not contain either $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$. So, perform $\langle T_i, \text{start} \rangle$ you will know that it has started, but because of failure it could not complete, because if it could complete or if before that if it had to roll back because of the normal execution then it would have written $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$, but because of system failure you could not write any one of them. So, the transaction has to be rolled back.

The other case is the case where the transaction needs to be redone is when then it contains the record $\langle T_i, \text{start} \rangle$, but in addition it also contains the record $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$. So, this is the transaction which had completed successfully, did the start, it did the commit or it rolled back the whole thing happened successfully, but because of system failure changes have not been able to take place and therefore, you will have to again execute that transactions. So, that is why you do a redo. In the earlier case it is undone you want to undo the effect, here the effects were given, but they somehow could not be made durable the database have become inconsistent. So, you need to redo that whole thing.

So, ma it may sound little bit awkwardness that if the it contains the $\langle T_i, \text{abort} \rangle$ why should you actually redo the transaction this is a just to keep things simple so that you

can just trace back the original history. So, you do not try to really optimize, but you just trace back the original history and do whatever had happened in the way and then that simplifies your algorithm significantly. And, then if there is a certain things which have been done by the undo operation you also want to go through those and maintain that status.

(Refer Slide Time: 33:23)

Immediate Modification Recovery Example

Below we show the log as it appears at three instances of time.

	(a)	(b)	(c)
<small>SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018</small>	$\langle T_0 \text{ start} \rangle$ $\langle T_0, A, 1000, 950 \rangle$ $\langle T_0, B, 2000, 2050 \rangle$ $\langle T_0 \text{ commit} \rangle$ $\langle T_1 \text{ start} \rangle$ $\langle T_1, C, 700, 600 \rangle$ $\langle T_1 \text{ commit} \rangle$	$\langle T_0 \text{ start} \rangle$ $\langle T_0, A, 1000, 950 \rangle$ $\langle T_0, B, 2000, 2050 \rangle$ $\langle T_0 \text{ commit} \rangle$ $\langle T_1 \text{ start} \rangle$ $\langle T_1, C, 700, 600 \rangle$ $\langle T_1 \text{ commit} \rangle$	$\langle T_0 \text{ start} \rangle$ $\langle T_0, A, 1000, 950 \rangle$ $\langle T_0, B, 2000, 2050 \rangle$ $\langle T_0 \text{ commit} \rangle$ $\langle T_1 \text{ start} \rangle$ $\langle T_1, C, 700, 600 \rangle$ $\langle T_1 \text{ commit} \rangle$

Recovery actions in each case above are:

- (a) undo (T_0): B is restored to 2000 and A to 1000, and log records $\langle T_0, B, 2000 \rangle$, $\langle T_0, A, 1000 \rangle$, $\langle T_0, \text{abort} \rangle$ are written out
- (b) redo (T_0) and undo (T_1): A and B are set to 950 and 2050 and C is restored to 700. Log records $\langle T_1, C, 700 \rangle$, $\langle T_1, \text{abort} \rangle$ are written out
- (c) redo (T_0) and redo (T_1): A and B are set to 950 and 2050 respectively. Then C is set to 600

Database System Concepts - 8th Edition 36.27 ©Silberschatz, Korth and Sudarshan

So, here are some examples here they transaction we are showing it is failure recovery action at every case. So, if in case a the transaction has started and we made changes to A and B and at that time the failure happens. So, naturally the start is there and at commit is not there or abort is no there. So, these has to be undone. So, this will be undone A will get back the value thousand and B we will get back the value 2000 and to such record $\langle T_0, B, 2000 \rangle$ and $\langle T_0, A, 2000 \rangle$ that two compensation log records will be and then it $\langle T_0, \text{abort} \rangle$ will be written.

Now, if you look at second transaction transaction just a second states of as in b then you will see that T_0 has actually started and committed and T_1 has started after that which could not complete after updating C. So, in the case of b since T_0 has start and commit both you have to redo that because you have lost all these changes we have to redo again and for that you do not log anything and then T_1 could not complete because it has start and does not have the abort or commit. So, you would log record for undoing it, undoing T_1 and you write $\langle T_1, C, 100 \rangle$ and $\langle T_1, \text{abort} \rangle$.

In the third case ma both transaction T 0 and transaction T 1 has commit start and commit and both have completed. So, you have to redo both of them. So, these are the basic different cases strategies that you have in place.

(Refer Slide Time: 34:59)

The slide has a header 'Checkpoints' with a small sailboat icon. The content lists several points about checkpoints:

- Redoing/undoing all transactions recorded in the log can be very slow
 - Processing the entire log is time-consuming if the system has run for a long time
 - We might unnecessarily redo transactions which have already output their updates to the database
- Streamline recovery procedure by periodically performing **checkpointing**
- All updates are stopped while doing checkpointing
 1. Output all log records currently residing in main memory onto stable storage
 2. Output all modified buffer blocks to the disk
 3. Write a log record <checkpoint L> onto stable storage where L is a list of all transactions active at the time of checkpoint

Navigation icons and slide details are visible at the bottom.

Now, the question is if you have to do this for all transactions when a failure happens and a failure may have happened say after 1 year or after 8, 9, 10 months and so on. So, there will be a huge you know set of redo, undo operations that you will have do it will run for a very long time. So, what we do is we create something like a check pointing where we said, ok. We will periodically choose a point of time where we will make sure that all updates have actually been consistently put in the disk and the database is surely on a consistent state and that is called check pointing.

So, whatever is done is a at a chosen point of check pointing, time of check pointing all updates are stopped in database. So, there is no changing changes happening in all transaction are no new transactions are allowed what is happening as.

So, you make sure that all records that are currently residing in your buffer is flushed on to the stable storage all modified buffer blocks which were not output we have also outputted and then you write that this is a write a log record saying the check point L on to the stable storage, where L is basically the transactions that were active at the time of checkpoint in the transactions that have already completed you do not need to remember because they have they are changes by the process of outputting all log records and all

modified buffer on to disk, you make sure that all completed transactions are fully secured now, they are consistent, they are you would have to, but those which are which were still continuing you keep the list and write that out in terms of the checkpoint log record and take it into the stable storage.

(Refer Slide Time: 36:52)

The slide has a decorative header image of a sailboat on water. The title 'Checkpoints (Cont.)' is centered in red. The content is a bulleted list under two main points:

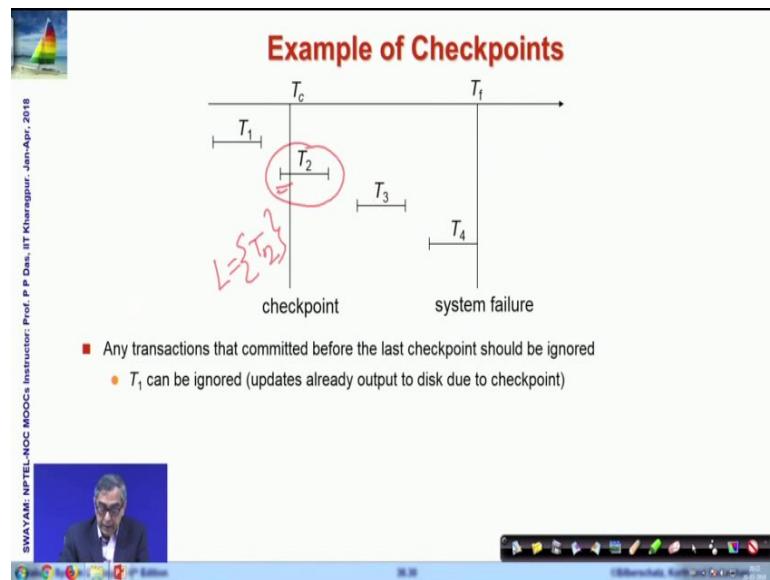
- During recovery we need to consider only the most recent transaction T_i that started before the checkpoint, and transactions that started after T_i
 - Scan backwards from end of log to find the most recent <checkpoint L> record
 - Only transactions that are in L or started after the checkpoint need to be redone or undone
 - Transactions that committed or aborted before the checkpoint already have all their updates output to stable storage
- Some earlier part of the log may be needed for undo operations
 - Continue scanning backwards till a record < T_i , start> is found for every transaction T_i in L
 - Parts of log prior to earliest < T_i , start> record above are not needed for recovery, and can be erased whenever desired

At the bottom left is a video frame showing a man speaking. On the right is a navigation bar with icons. The footer contains text: 'SWAYAM-NIPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8th Edition', '36.29', and '©Silberschatz, Korth and Sudarshan'.

So, during recovery what we need to consider is we have to now we not have to go back to the last time the disk fail we just need to go back to the last time we did check pointing and when you go back to the check pointing you already know that ma what are the transactions that are live at the time of check pointing. So, you can scan backwards and check out what were they had started. So, you need to and undo redo those are transactions and then you see what are the transactions that have committed aborted have already there in the output in the stable storage.

So, some of the earlier part of the log may need may be needed for undo operations. So, you continue scanning backwards till you find in < T_i , start> and then you take care of that.

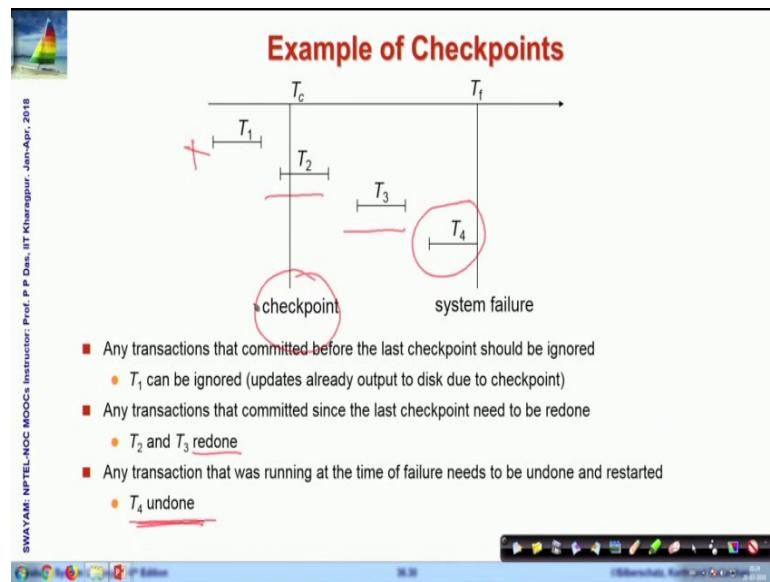
(Refer Slide Time: 37:48)



Let me just explain through an example. So, let us say that this is the checkpoint where you froze everything and did not allow any further updates to happen and rolled back all the data. So, what has happened is in this transaction T_1 which is committed before the last checkpoint naturally there was no obtained pending for that. So, you have made sure that all the updates in terms of the log as well as the system buffer has been written on to have been output on to the disk at the time of check pointing. So, you do not remember need to remember this transaction at all, so this can simply be ignored.

Now, at the checkpoint you can see that transaction T_1 was in execution. So, certain things had happened. So, at the checkpoint the part that has already happened the log records for that as well as the output for that this has already been firmly put into that because these are checkpoints because you are writing everything, but this transaction is still in execution. So, you will put this transaction in the checkpoint log list. So, we will say this is this is the T_2 and this will need to be looked at. If you so, let us see what is you will do with this.

(Refer Slide Time: 39:02)

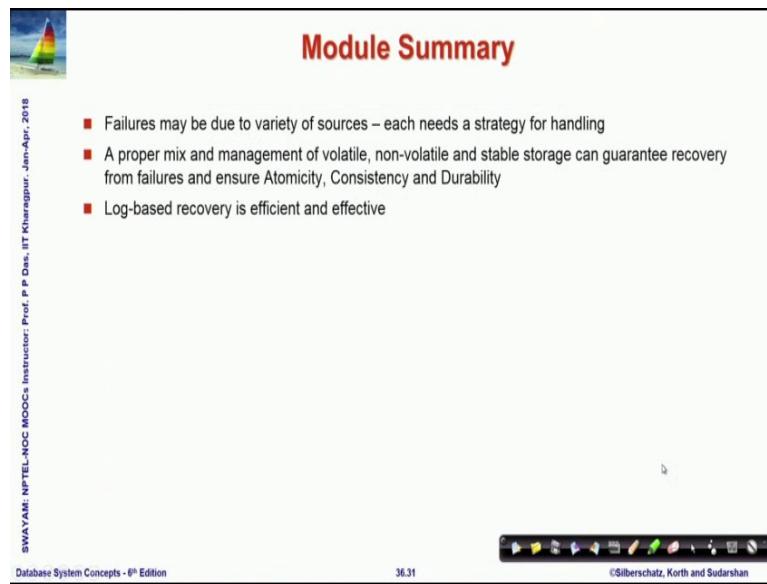


So, if we look at then naturally T_2 if you have a failure at this point if you have a failure at this point as you have here then naturally this is the last stable point you know where everything was written to the disk in a consistent manner. So, what you will need to do you will have to execute transaction T_2 once more to make sure that it is you know up to this point this being done, so, you need to redo this part. Similarly, T_3 if you look at it started after the checkpoint and it committed before the system failure. So, you need to redo that as well.

And, if you look into the T_4 if you look into T_4 you can see that it started before the system failure of course, after the checkpoint and it was still running when the failure happens. So, you do not know what are the final results of that, so, what you will need to do? You will need to undo this transaction. So, this has no impact you can just ignore these cases you have to redo the transaction and in this case, in case of T_4 you have to undo the transaction.

So, by check pointing and you obviously, the point you choose for check pointing has to be judiciously done it may not be very frequent and then it will there be a lot of overhead at the same time if you do it in a very long period of time then naturally you will not get the benefits, but check pointing is a very critical feature of doing the recovery in the databases.

(Refer Slide Time: 40:36)



The slide has a header 'Module Summary' with a sailboat icon. It contains a bulleted list of three items. The footer includes course details, a navigation bar, and copyright information.

Module Summary

- Failures may be due to variety of sources – each needs a strategy for handling
- A proper mix and management of volatile, non-volatile and stable storage can guarantee recovery from failures and ensure Atomicity, Consistency and Durability
- Log-based recovery is efficient and effective

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition 36.31 ©Silberschatz, Korth and Sudarshan

So, to summarize we have seen that there are many different types of failures and different strategies are required for handling them. And we have also seen that we use different kinds of storage structures and their judicious mix and then arrangement of these structures can guarantee recovery from failures. And we have taken a brief look into the log base recovery mechanism which is efficient as well as effective and I will remind you that all this discussion was done for serial transactions only.

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

Lecture – 37
Recovery/2

Welcome to module 37 of Database Management Systems. We have been discussing about Database Recovery. This is the second and concluding part of the Database Recovery.

(Refer Slide Time: 00:25)

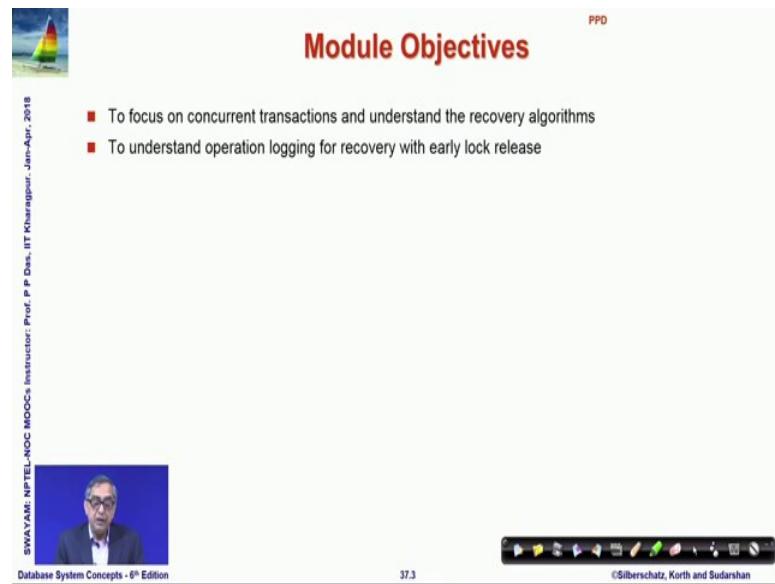
The slide is titled "Module Recap" in red at the top right. On the left, there is a small image of a sailboat on water. The footer contains copyright information: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P P Das, IIT Kharagpur", "Database System Concepts - 8th Edition", "37.2", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the bottom right.

Module Recap

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

We have earlier discussed about failure classification, storage structures and significantly the log based recovery mechanism.

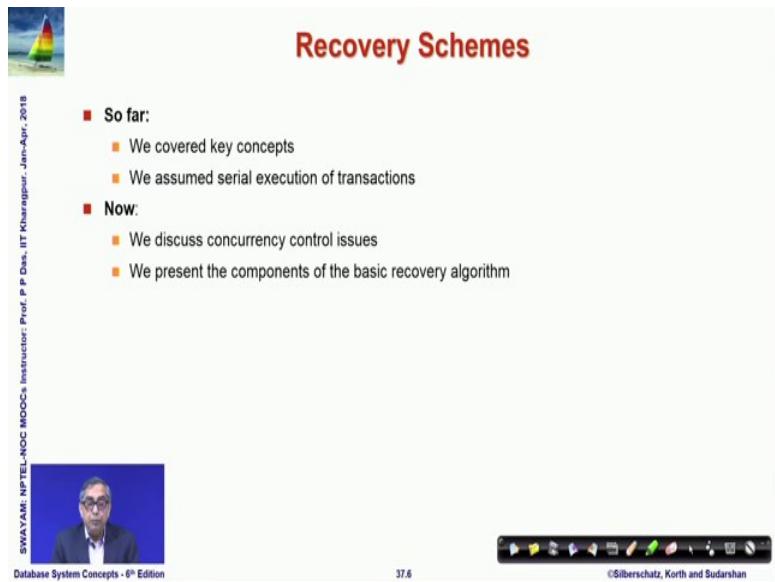
(Refer Slide Time: 00:34)



The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Jan-Apr, 2018". A video thumbnail of the instructor is in the center-left. The main content area contains two bullet points: "To focus on concurrent transactions and understand the recovery algorithms" and "To understand operation logging for recovery with early lock release". At the bottom, it says "Database System Concepts - 6th Edition", "37.3", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom right.

In this, we will focus on concurrent transactions and understand the recovery algorithm for them and we will understand the operation of logging for recovery with early lock release. We will learn about another kind of logging mechanism; so, the recovery algorithm.

(Refer Slide Time: 00:51)



The slide is titled "Recovery Schemes" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Jan-Apr, 2018". A video thumbnail of the instructor is in the center-left. The main content area contains two sections: "So far:" with points "We covered key concepts" and "We assumed serial execution of transactions"; and "Now:" with points "We discuss concurrency control issues" and "We present the components of the basic recovery algorithm". At the bottom, it says "Database System Concepts - 6th Edition", "37.6", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom right.

So, what we have seen so far are we have learned the basic concept of recovery and logging and we have assumed the serial execution of transactions and now we discussed

the Concurrency Control issues. So, now, we will assume that there are multiple transactions operating at the same time and the components that are required for the recovery of those.

(Refer Slide Time: 01:10)

The slide has a title 'Concurrency Control and Recovery' in red. To the left of the title is a small icon of a sailboat on water. Below the title is a video player window showing a man in a suit speaking. The video player has a progress bar and several control icons. On the left side of the slide, there is vertical text that reads 'SVAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8th Edition'. At the bottom right is the text '©Silberschatz, Korth and Sudarshan'.

- With concurrent transactions, all transactions share a single disk buffer and a single log
 - A buffer block can have data items updated by one or more transactions
- We assume that *if a transaction T_i has modified an item, no other transaction can modify the same item until T_i has committed or aborted*
 - That is, the updates of uncommitted transactions should not be visible to other transactions
 - ▶ Otherwise how do we perform undo if T_1 updates A, then T_2 updates A and commits, and finally T_1 has to abort?
 - Can be ensured by obtaining exclusive locks on updated items and holding the locks till end of transaction (strict two-phase locking)
- Log records of different transactions may be interspersed in the log

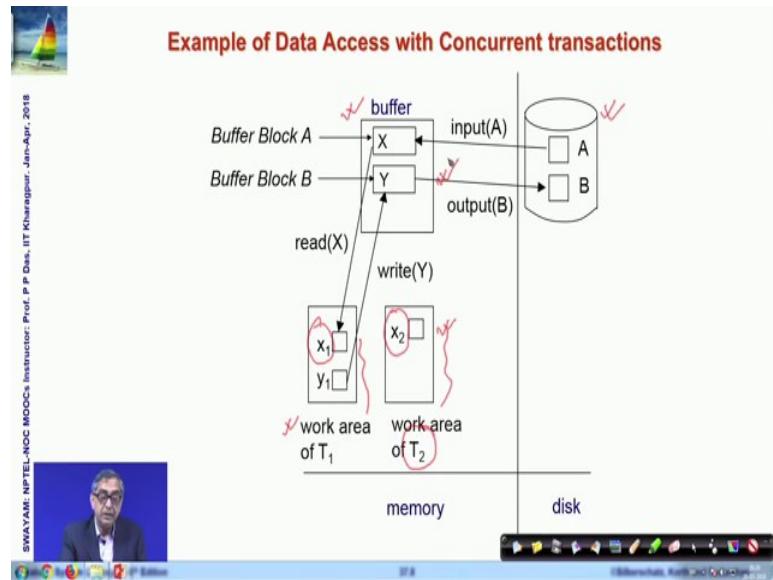
So, with Concurrent Transaction, all transactions share we already know that every transaction is a private work area that assumption stays, but we talked about a system buffer area.

So, that system buffer area the that area would be common for all different transactions, also the log area would be common for all transactions. So, now, the in the buffer area the, data rights are or reads or writes are done for different transactions and in the log the different logs of different transactions are fixed up. So, we make one assumption that if a transaction has modified an item, then no other transaction can modify that same item unless that transaction is committed or aborted.

So, which means that kind of when the transaction modifies the item it holds a lock and that lock is held till the end of the transaction and this is a I mean if we if we think back in terms of our locking protocol, this is a strict locking protocol that we are talking of. This is important for recovery because if we did not have this, then it is possible that multiple updates to the same rate item are done by multiple transactions. So, if we have

to undo that then we will not know we were which one it to be undone with. So, that is the basic problem. So, we will assume an exclusive lock in this case and log records will be written interspersed as we have already saw.

(Refer Slide Time: 02:43)



So, in terms of our storage access mechanism, the same as earlier diagram, this is here is a disk, here is a buffer, the buffer is common and the private work area. So, now, in addition to T_1 , we have another transaction T_2 with its own buffer area, but so, the x has been written in T_1 , has been read in T_1 as x_1 , x has also been read in T_2 as x_2 and each are concurrently making changes in that private work area, but they are using the same system buffer area for writing the output back to the disk or reading directly from the disk. So, this is a model that we will go with.

(Refer Slide Time: 03:30)

The slide features a title 'Recovery Algorithm' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading '■ Logging (during normal operation):':

- $\langle T_i \text{ start} \rangle$ at transaction start
- $\langle T_i, X_p, V_1, V_2 \rangle$ for each update, and
- $\langle T_i \text{ commit} \rangle$ at transaction end

At the bottom left, there is a video thumbnail showing a man speaking. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '37.9', and '©Silberschatz, Korth and Sudarshan'.

So, what is the recovery algorithm, first is logging and the logging structure remains same; the start transaction log, the update transaction log and the commit transaction log as before.

(Refer Slide Time: 03:43)

The slide features a title 'Recovery Algorithm (Contd.)' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading '■ Transaction rollback (during normal operation)':

- Let T_i be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of T_i of the form $\langle T_i, X_p, V_1, V_2 \rangle$
 - perform the undo by writing V_1 to X_p
 - write a log record $\langle T_i, X_p, V_1 \rangle$
 - such log records are called **compensation log records**
- Once the record $\langle T_i \text{ start} \rangle$ is found stop the scan and write the log record $\langle T_i \text{ abort} \rangle$

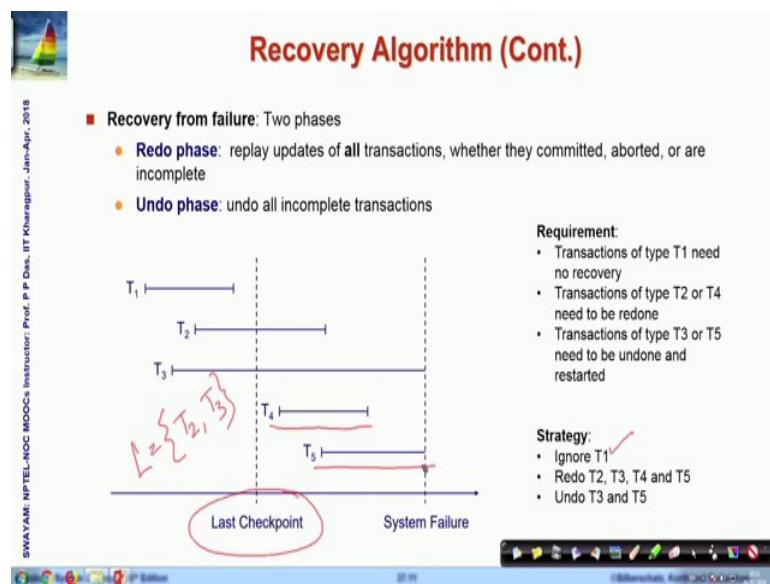
At the bottom left, there is a video thumbnail showing a man speaking. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '37.10', and '©Silberschatz, Korth and Sudarshan'.

When you have to do a transaction rollback during normal operation; so, for that transaction T_i to be rolled back, what we will need to do it is a rollback. So, undo has to happen. So, scan will scan the log backwards from the end and for each log record

update log record, we will restore the original value for which was written over and we will write a compensation log record as before and going backwards in this way when we come across the start log record, then we will stop that scan and write a abort log record in that place.

So, it is exactly same to what we did.

(Refer Slide Time: 04:21)



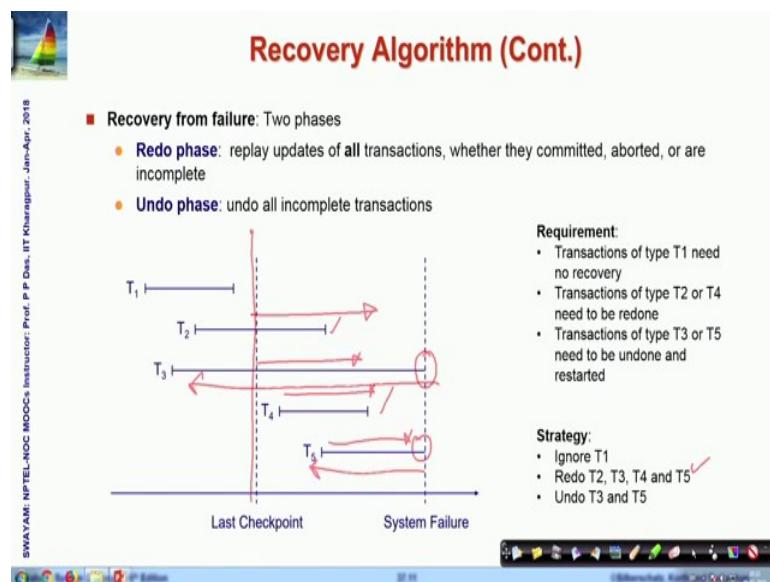
So, now let us look into the actual Recovery Algorithm. So, the transaction rollback has no difference. So, in the Recovery Algorithm, what we do we have a recovery phase and where we replay updates of all transactions. So, we make sure that all transactions whatever they did they those are done again. So, after the failure we recover from the failure. So, we up do all that again whether they are committed, whether they are aborted, whether they are incomplete in every case and then we keep track of what are the transactions which did not complete and for them we do an undo phase. So, here I am showing another example here.

So, this is the last checkpoint where eh all updates I mean freezing the updates, everything was output to the disk the log as well as the data item updates were put to the disk and the set of transactions that are live during that time well execution in that time

were recorded. So, if we look at that set L in this case, then it will be T_2 , T_3 these two transactions.

So, we can we have already seen that our strategy would be that we will ignore T_1 because it had completed before the last checkpoint. T_2 and T_3 were ongoing and then T_4 has started after checkpoint and committed before that, T_5 stared after checkpoint, but was also active was also in execution when system failed. So, our strategy would be, that we will assume as if this, this whole thing as is redone.

(Refer Slide Time: 06:06)



So, T_2 , T_3 , T_4 , T_5 all these log records exist. So, we will follow through them and redo all of them. If we redo all of them then naturally we come across T_3 and T_5 which cannot proceed further because the system had could not proceed further because. So, we do not know in terms of the log what would have happened to them because the system had failed.

So, after having done this then, we do an undo phase where we undo this, but naturally the effect of these will remain. Now you can question that this could have been done in a more smart way, do we really need to redo everything and then undo some parts of that, that is a override in terms of that which is true, but this just makes the whole algorithm simple and over it actually is not very hard.

(Refer Slide Time: 06:55)

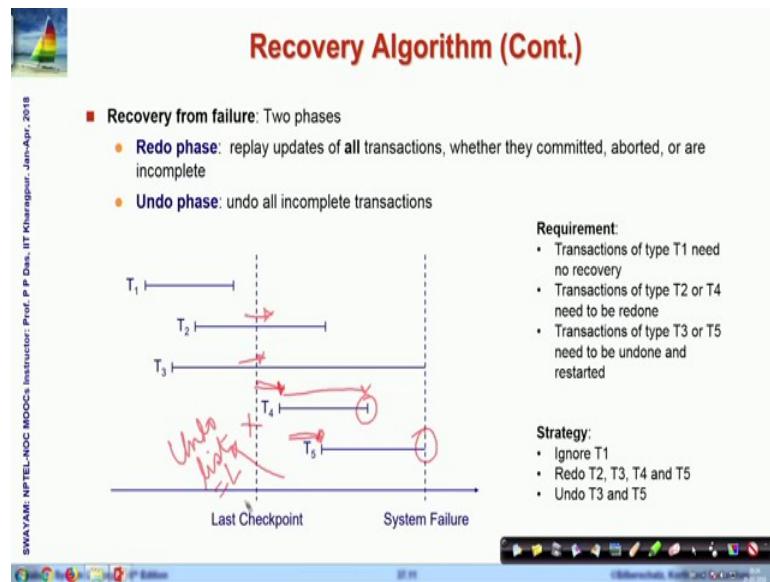
The slide has a header 'Recovery Algorithm (Cont.)' with a sailboat icon. The main content is a bulleted list under 'Redo phase':

- 1. Find last <checkpoint L> record, and set undo-list to L
- 2. Scan forward from above <checkpoint L> record
 - 1. Whenever a record < $T_j X_j V_1 V_2$ > is found, redo it by writing V_2 to X_j
 - 2. Whenever a log record < T_i start> is found, add T_i to undo-list
 - 3. Whenever a log record < T_i commit> or < T_i abort> is found, remove T_i from undo-list

Navigation icons and text at the bottom include: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018; Database System Concepts - 8th Edition; 37.12; ©Silberschatz, Korth and Sudarshan.

So, we are doing the redo phase, even the redo phase you will find the check point and you will scan forward from the checkpoint record and as you scan forward from the checkpoint record; if you have an update, you will simply redo which means V_2 , will again be written to X_j and when you find a start transaction, then you do not know. Just look at this point carefully; if you find a start transaction for example, when you are working on this, suppose you come across a start transaction here, you will come across the start transaction transactions start here.

(Refer Slide Time: 07:31)



So, whenever you get that, then you put this into the undo list. Initially your undo list is L because they were going on. So, you do not know that they could finish all that still need to be undone and when you come across a new start, you add that to the undo list and then the rest of it is simple. So, you keep on going this way, if you find that the commit has happened or abort has happened, you remove that from the undo list, but if you do not find that then that stays in the undo list. So, you know if you, if you proceed from in this direction in the redo phase, you know and that way when you have scanned the whole log, you know what are the transactions which still need to be undone. So, that is a simple strategy that is followed.

(Refer Slide Time: 08:26)

The slide features a small sailboat icon in the top left corner. The title 'Recovery Algorithm (Cont.)' is centered at the top in red. Below the title, a section header '■ Redo phase:' is followed by a numbered list:

1. Find last <checkpoint L> record, and set undo-list to L
2. Scan forward from above <checkpoint L> record
 1. Whenever a record < T_i, X_j, V_1, V_2 > is found, redo it by writing V_2 to X_j
 2. Whenever a log record < T_i, start > is found, add T_i to undo-list
 3. Whenever a log record < T_i, commit > or < T_i, abort > is found, remove T_i from undo-list

On the right side of the slide, there is a watermark-like text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls.

So, ma whenever you have a log record start, then you put it to the undo list and whenever you get a log record which is committed abort which says that before the system failure the transaction actually had either committed that it finished everything or you had to roll back, then you remove that from the undo list. So, what will be left out, at the end will be the undo list of transactions that need to be undone subsequently.

(Refer Slide Time: 08:53)

The slide features a small sailboat icon in the top left corner. The title 'Recovery Algorithm (Cont.)' is centered at the top in red. Below the title, a section header '■ Undo phase:' is followed by a numbered list:

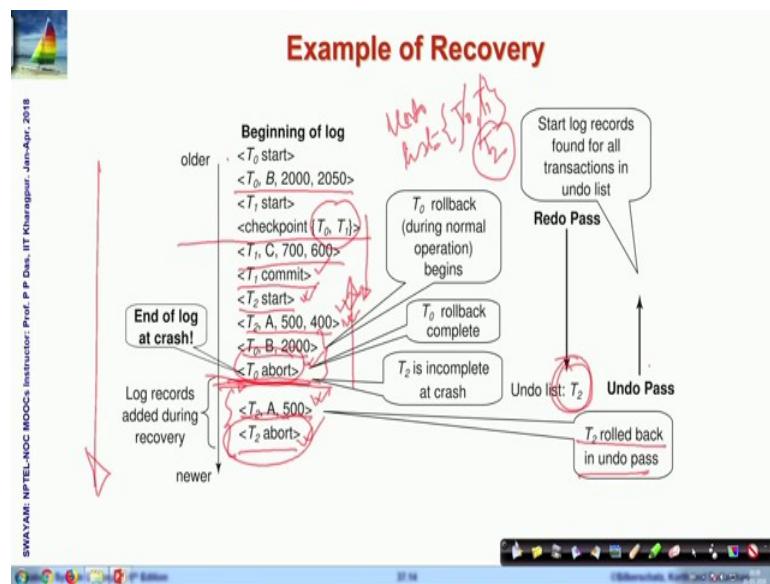
1. Scan log backwards from end
 1. Whenever a log record < T_i, X_j, V_1, V_2 > is found where T_i is in undo-list perform same actions as for transaction rollback:
 1. Perform undo by writing V_1 to X_j
 2. Write a log record < T_i, X_j, V_1 >
 2. Whenever a log record < T_i, start > is found where T_i is in undo-list,
 1. Write a log record < T_i, abort >
 2. Remove T_i from undo-list
 3. Stop when undo-list is empty
 - That is, < T_i, start > has been found for every transaction in undo-list
 - After undo phase completes, normal transaction processing can commence

On the right side of the slide, there is a watermark-like text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom, there is a video player showing a video of a professor and a navigation bar with icons for back, forward, search, and other presentation controls.

In the undo phase, in the undo phase, you go backwards because it is undo. So, what you will do is a very similar. So, if you have an update record, then you undo in the main transaction which is in undo list then you do exactly in terms of transaction rollback that you write the old value and write a redo only log record. Now when you find going backwards, when you find $\langle T_i, \text{start} \rangle$; so you know that this is a starting point of the transaction and the transaction is in undo list. So, it came across because it could not be it was on the undo list. So, which means that it could not be completed and therefore, you have found the start. So, this is where your undo operation is over. So, you write a abort log record and once you have written that, then you are done with the transaction. So, you remove that from the undo list and in this way, you will continue till your undo list is becomes empty.

Once it becomes empty, so, then you have found that $\langle T_i, \text{start} \rangle$ for all transactions in the undo list and there is nothing more to do. So, after undo phase completes normal transaction processing can comment. So, your failure recovery from the failure is already taken care of.

(Refer Slide Time: 10:14)



So, here are certain examples which you could check out, here are a start. So, this is the how that this is the order in which the transactions are going and this is the crash point, these is where it failed and mind you. So, this is where and this is where our checkpoint

is. So, at checkpoint you can see that T_0 and T_1 are; what are your candidates? So, when you start in the redo phase, you start from this point because before that everything has been done. You naturally, you come across this. So, you redo this which means you again actually change C from its old value 700 to 600 and then T_1 commits. As T_1 commits, you know that this transaction is done with.

So, you remove that. So, your undo list at the beginning is T_0, T_1 , but going in the forward direction when you come across **<T₁, commit>**, you naturally from your undo list, then you come across **<T₂ , start>**. So, you know that another transaction is starting now. So, it may be you do not know whether it could complete or you could not. So, you add that to the undo list then give effect to this update, then if for T_0 we have you have a rollback record that is because T_0 actually you can see that T_0 has aborted. So, the change that T_0 had done earlier this had to be rolled back, this rollback is a normal transaction rollback, this is not because of the failure. So, this mm rollback had happened and this is where the rollback is complete.

So, T_2 e, T_0 is also completed. So, T_0 after this is taken care of, then in the redo phase T_0 is also complete and this is where you reach the crash point. So, your undo list is left with only T_2 . So, now, when you have done this, so when you have taken done the redo here that T_2 which is ongoing is there, then you write this log record. So, these log records are written during recovery not during the original transaction and T_2 had to abort because of the system failure.

So, T_0 support was due to the transaction rollback, but **<T₂ , abort>** is because of the system failure. So, T_2 is rolled back in the redo phase. So, once this has been done, then you do the undo phase starting with T_2 and then you go backwards as you go backwards here. So, you will undo this, this is what you write you come across T_2 and naturally you have rollback. So, you write **<T₂, abort>**. This is how the actual rollback can happen and you can see that now the with this redo undo phase you can always bring back the database to a consistent state and these transactions are executing concurrently and therefore, your log record is a intermix of the log record of different transactions. Now the last that we would like to talk about is Recovery with Early Lock Release.

(Refer Slide Time: 13:55)

The slide has a header 'Recovery with Early Lock Release' with a sailboat icon. On the left, there's vertical text: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Deshpande', 'IIT Kharagpur', and 'Jan-Apr., 2018'. The main content is a bulleted list:

- Support for high-concurrency locking techniques, such as those used for B*-tree concurrency control, which release locks early
 - Supports "logical undo"
- Recovery based on "repeating history", whereby recovery executes exactly the same actions as normal processing

At the bottom left is a video player interface showing a man speaking, with the text 'Database System Concepts - 8th Edition'. The bottom right shows slide navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

What this means is well, so far we have talked about recovery which is only in terms of data update, single data updates. So, I mean if I want to recover I can just you know write back the old data, but this is not true in case of many other situations for example, if you are inserting a record in a B-tree, then it is not enough only to undo that because you cannot undo and get back the same.

As you can understand that if you make inserts or deletes in the B-tree, if you are made an insert then the structure of the B-tree itself has changed and after that several other inserts, deletes may have happened. So, if you now want to just go back and undo this particular insert in terms of values, it is not possible to do that. So, when you want to do that, so you cannot do really kind of repeating the history kind of strategy.

(Refer Slide Time: 14:53)

The slide has a header 'Logical Undo Logging' with a sailboat icon. The main content is a bulleted list under a red square icon:

- Operations like B+-tree insertions and deletions release locks early
 - They cannot be undone by restoring old values (**physical undo**), since once a lock is released, other transactions may have updated the B+-tree
 - Instead, insertions (resp. deletions) are undone by executing a deletion (resp. insertion) operation (known as **logical undo**)
- For such operations, undo log records should contain the undo operation to be executed
 - Such logging is called **logical undo logging**, in contrast to **physical undo logging**
 - ▶ Operations are called **logical operations**
 - Other examples:
 - ▶ delete of tuple, to undo insert of tuple
 - allows early lock release on space allocation information
 - ▶ subtract amount deposited, to undo deposit
 - allows early lock release on bank balance

At the bottom left is a video thumbnail of a professor, at the bottom center is the page number '37.17', and at the bottom right is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, what you have to do is do some kind of an undo which is logical. So, so far the undo was physical that, you wrote this, you change this value by this value. So, your undo is a physical. So, you restore the original value and your undo is done here. It is logical that is for the operation that you have performed, you try to find out a matching operation which creates the similar effect as of undo. So, if you have inserted, then your undo is a corresponding delete of that record. If you have incremented by 10 then you can say that your corresponding undo is a decrement by 10. So, that is what is known as the logical undo and it is logical undo is a very good option in case of delete of, insert delete of people.

So, if you have deleted a people undo to insert, if you have subtracted then undo to undo deposit to go forward and so on.

(Refer Slide Time: 15:55)

The slide has a header 'Physical Redo' with a sailboat icon. The main content lists advantages of physical redo:

- Redo information is logged **physically** (that is, new value for each write) even for operations with logical undo
 - Logical redo is very complicated since database state on disk may not be "operation consistent" when recovery starts
 - Physical redo logging does not conflict with early lock release

Navigation icons and slide details are at the bottom.

So, a redo information is logged physically, so new values for each right even for operations which are logically, which has logical undo. So, you do not do a logical redo I mean, I will not go into the details of why this is not done, but it simply makes things very complicated. So, physical redo is always physical and you can show that physical redo does not prohibit this kind of operations that we are trying to do, but the logical redo is not used. We will only use logical undo operation.

(Refer Slide Time: 16:40)

The slide has a header 'Operation Logging' with a sailboat icon. It describes the process of operation logging:

- Operation logging is done as follows:
 - When operation starts, log $\langle T_p, O_p, \text{operation-begin} \rangle$. Here O_p is a unique identifier of the operation instance
 - While operation is executing, normal log records with physical redo and physical undo information are logged
 - When operation completes, $\langle T_p, O_p, \text{operation-end}, U \rangle$ is logged, where U contains information needed to perform a logical undo operation

Example: insert of (key, record-id) pair (K5, RID7) into index I9

Log sequence:
<T1, O1, operation-begin>
....
<T1, X, 10, K5>
<T1, Y, 45, RID7>
<T1, O1, operation-end, (delete I9, K5, RID7)>

Navigation icons and slide details are at the bottom.

So, how do you log for such a logical undo operation, what you do is instead of now. So, now, it is an operation it may not be a single value update. So, it is not captured in terms of one you know log record, but it could be a number of log records which have actually done three, four different changes to make that operation happen and you want to actually define an undo for that operation. So, when you start this. So, you start with a log which says that what is the transaction and what is the operation. So, you put an identifier to the operation and then you write operation begin.

So, you know this is where operation has started, then all the things that are happening for this operation while the operation is executing then you write normal log records with physical redo physical information. All these logs are written and when this operation ends mind you, this is a particular operation you are talking of. So, not the whole transaction whole transaction will continue when that particular operation ends, then you write an operation in record and along with that you write, what is a logical, what is a logical undo information you put that in.

So, let us have a look at the example. So, suppose your operation is insert of a key record pair, so, let us say this is the key record pair and into index I9. So, this operation starts here and then there are several steps to be done; for example, you will have to say if X is on the key value which had 10 and is becoming K5, you will have a physical update undo record of this. If Y is the record id which is RID 7, then it Y changes from 45 to. So, these are all physical redo steps in insert. So, these are the different instructions in terms of this broad operation and when you are done with all that then your operation ends and you write this undo information. So, insert of, so you had insert of this record with index 9.

So, now you do write your what will be the undo, to delete that from index I9, to delete this key record ID pair. So, this is a whole locking that we do. So, you can make use of this undo operation in terms of your recovery process.

(Refer Slide Time: 19:22)

The slide has a header 'Operation Logging (Cont.)' with a sailboat icon. The main content is a bulleted list:

- If crash/rollback occurs before operation completes:
 - the **operation-end** log record is not found, and
 - the physical undo information is used to undo operation
- If crash/rollback occurs after the operation completes:
 - the **operation-end** log record is found, and in this case
 - logical undo is performed using U ; the physical undo information for the operation is ignored
- Redo of operation (after crash) still uses physical redo information

Navigation icons at the bottom include arrows, a magnifying glass, and other symbols. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '37.20', and '©Silberschatz, Korth and Sudarshan'.

So, if the crash or rollback occurs before the operation completes, then operation and log record is not found you will not find it. So, you do not know what is the undo operation. So, in that case the physical undo information is will be used to undo, but if we have a crash on rollback that happens after an operation completes, then the operation end log will be available and in this case we will use the undo operation that is given in the operation end log record and do a logical undo. And we will ignore all the physical undo information that the operation that that we will find in the log records. Redo of course will still use the physical redo information which is there.

(Refer Slide Time: 20:10)

Transaction Rollback with Logical Undo

Rollback of transaction T_i , scan the log backwards

1. If a log record $\langle T_p, X, V_1, V_2 \rangle$ is found, perform the undo and log $\langle T_p, X, V_1 \rangle$
2. If a $\langle T_p, O_p, \text{operation-end}, U \rangle$ record is found
 - Rollback the operation logically using the undo information U
 - Updates performed during roll back are logged just like during normal operation execution
 - At the end of the operation rollback, instead of logging an **operation-end** record, generate a record $\langle T_p, O_p, \text{operation-abort} \rangle$
 - Skip all preceding log records for T_i until the record $\langle T_p, O_p, \text{operation-begin} \rangle$ is found
3. If a redo-only record is found ignore it
4. If a $\langle T_p, O_p, \text{operation-abort} \rangle$ record is found:
 - skip all preceding log records for T_i until the record $\langle T_p, O_p, \text{operation-begin} \rangle$ is found
5. Stop the scan when the record $\langle T_p, \text{start} \rangle$ is found
6. Add a $\langle T_p, \text{abort} \rangle$ record to the log

Note:

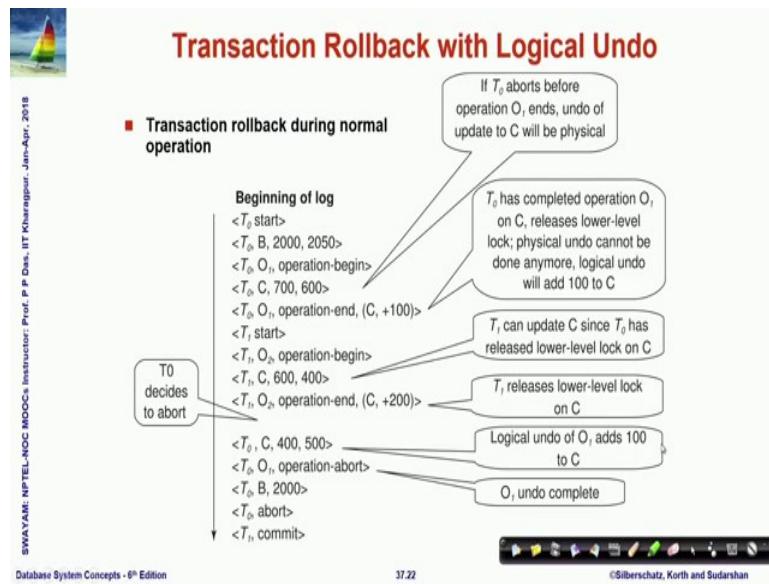
- Cases 3 and 4 above can occur only if the database crashes while a transaction is being rolled back
- Skipping of log records as in case 4 is important to prevent multiple rollback of the same operation

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018
Database System Concepts - 8th Edition 37.21 ©Silberschatz, Korth and Sudarshan

So, if we look into the actual if we if we look into the transaction rollback with logical undo. So, if I have an update record which is naturally physical, and then we can perform the undo which is as we did last time the creating a redo only record. If I find an operation end record, then to rollback we will pick up the logical undo information from you and we will perform that operation. At the end of that we will certainly write the operation abort record to show to mark that this operation has been aborted.

So, if we have a redo only record, then we will ignore it and if we find an operation abort, then we will skip all the records that were found till the beginning. Naturally, you can you can understand that 3 and 4 will not happen in a normal course of transaction, it will happen only when the failures have happened during recovery and at the end we will add $\langle T_i, \text{abort} \rangle$ record to the log. So, the critical thing to remember in this that whenever we are doing operation hmm unlogging, we are doing undoing based on the operation logging then since once we get the operation ends since we know what the undo information is, we have to make sure that through the undo process we actually ignore the physical undo records that exist in the log and just go with the operation case. So, this these are the notes I just mentioned it ok.

(Refer Slide Time: 22:15)



So, this is an example which you will have to spend some time and understand with care. So, you can see that a transaction T_0 has started, this is where it has done a physical update, is a physical undo record and then it does an operation. Of course, it is a simple operation which changes the value of C from 700 to 600. So, naturally, so it has decremented by 100. So, your undo operation here is incrementing by 100.

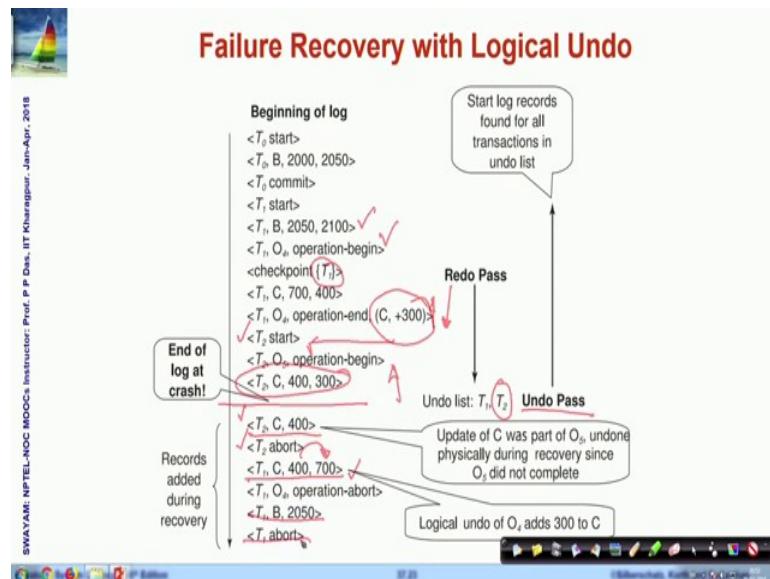
So, if T_0 aborts here, if T_0 aborts somewhere here you know before your operation end has happened then naturally the undo will have to be based on this physical undo structure. So, you will have to replace 600 by 700, but if it happens after this, then this is the case if it is completed the operation and then the failure happens, then you will do the logical undo that is whatever the value of C is you will just logically add 100 to that. But that means, that when you go backwards from here to find the begin, you will actually have to ignore this physical undo record because you have already given effect to that in terms of the operation undo that you are doing.

So, this is the basic difference. Here are different subsequent examples on that and you can see that well here after the operation end has happened, then possibly it has released the T_0 has released a lock on C_1 . So, T_1 has again taken the log. T_1 has again done the updates. So, then it releases that and T_0 at this point might decide to abort; if T_0 aborts, then this logical undo of O_1 this operation will add, it had to add 100. So, it adds

now this C had become 400, now it is adding 100 back. So, C becomes 500 and then the operation is finished. So, you write operation abort and O_1 undo of O_1 gets completed, but you still have after going backwards in this, you still have this record which was directly updated.

So, these are the undo transactions, undo lock record for that where B is being restored from 2050 to 2000 and you record the transaction abort for T_0 , T_1 eventually has committed here. So, this is how the transaction rollback will happen when logical undo is also used and this is a very powerful way to take care of that.

(Refer Slide Time: 25:13)



This is similarly another illustration for doing the failure recovery for with logical undo. So, here is the undo is a re redo phase that you are seeing here, this is where the end of log at the crash these are redo phase because these are check point where T_1 was there. So, at the end of redo T_1 if you if you. So, you are starting to redo from here. So, you have done operation end. T_1 has not finished T_2 has started. So, you have added T_2 to the undo list and when the crash has happened both of them are on the undo list. So, they have to go through that undo pass. So, we undo $<T_2, C, 400>$. So, this is what this is this is how you will go.

So, this is the first thing you undo and then naturally you have come to the beginning of **<T2 , start>**. So, you abort and you are going back again and you are trying to do this. Why are you doing this because when you go back to undo from this point you come across this operation end which tells you that the undo operation has to happen by incrementing C by 300. So, C which had become 400 is now incremented by 300. You come to the check point which is the end here in terms of the operation begin and naturally you declare operation abort and going back further this is what you had when transaction T₁ had started.

So, you undo that. That is again a physical undo and finally, T₁ aborts. So, this is how in both cases of transaction rollback as well as in terms of the failure the recovery can be done with the logical undo process.

(Refer Slide Time: 27:10)

Transaction Rollback: Another Example

- Example with a complete and an incomplete operation

```

<T1, start>
<T1, O1, operation-begin>
...
<T1, X, 10, K5>
<T1, Y, 45, RID7>
<T1, O1, operation-end, (delete I9, K5, RID7)>
<T1, O2, operation-begin>
<T1, Z, 45, 70>
    ← T1 Rollback begins here
<T1, Z, 45> ← redo-only log record during physical undo (of incomplete O2)
<T1, Y, ... ,> ← Normal redo records for logical undo of O1
...
<T1, O1, operation-abort> ← What if crash occurred immediately after this?
<T1, abort>

```

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr- 2018

Database System Concepts - 8th Edition 37.24 ©Silberschatz, Korth and Sudarshan

Here I have given another example. I will not go through it step by step. So, at a arts that you go through that following the same logic and convince yourself that you understand that how this transaction rollbacks with physical undo as well as logical undo is taking place.

(Refer Slide Time: 27:27)

The slide features a sailboat icon in the top left corner. The title 'Recovery Algorithm with Logical Undo' is centered at the top in a red header. Below the title, a vertical column of text provides background information and steps for the recovery algorithm.

Basically same as earlier algorithm, except for changes described earlier for transaction rollback

1. (Redo phase): Scan log forward from last <checkpoint L> record till end of log

1. Repeat history by physically redoing all updates of all transactions,
2. Create an undo-list during the scan as follows
 - undo-list is set to L initially
 - Whenever < T_i start> is found T_i is added to undo-list
 - Whenever < T_i commit> or < T_i abort> is found, T_i is deleted from undo-list

This brings database to state as of crash, with committed as well as uncommitted transactions having been redone

Now undo-list contains transactions that are incomplete, that is, have neither committed nor been fully rolled back

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

37.25

©Silberschatz, Korth and Sudarshan

So, ma with this Recovery Algorithm with logical undo will look very similar to what we have already done with the physical undo redo and though that is what we have stated here, there is no nothing significantly new. So, I expect that you should be able to go through these steps.

(Refer Slide Time: 27:52)

The slide features a sailboat icon in the top left corner. The title 'Recovery with Logical Undo (Cont.)' is centered at the top in a red header. Below the title, a vertical column of text continues the explanation of the recovery process.

Recovery from system crash (cont.)

2. (Undo phase): Scan log backwards, performing undo on log records of transactions found in undo-list

- Log records of transactions being rolled back are processed as described earlier, as they are found
 - Single shared scan for all transactions being undone
- When < T_i start> is found for a transaction T_i in undo-list, write a < T_i abort> log record.
- Stop scan when < T_i start> records have been found for all T_i in undo-list

■ This undoes the effects of incomplete transactions (those with neither commit nor abort log records). Recovery is now complete

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

37.26

©Silberschatz, Korth and Sudarshan

And those will be clear to you again we have a two phase recovery of redo phase and the undo phase and we make use of the operation undo, logical undo as and when it is

possible and when that is and when we do that, we ignore all physical undo records and when it is not possible, then we lose the physical undo records and that is how the recovery can be achieved.

(Refer Slide Time: 28:19)

The slide has a title 'Module Summary' in red at the top right. Below it is a list of two bullet points: 'Studies the recovery algorithms for concurrent transactions' and 'Recovery based on operation logging supplements log-based recovery'. On the left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom of the slide, there is footer text 'Database System Concepts - 8th Edition' and '37.27' next to a progress bar. To the right of the progress bar is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, in this module we have exposed ourselves with the Recovery Algorithms now for concurrent transactions as well and we have shown that how recovery can be done using operational logging, operations logging and making sure that really the database may not need to hold on to a lock for a long time on the data item and delay other transactions, but if it can define the undo operation on the data item, then it can release that log early and use that logging mechanism operation logging mechanism to recover the data.

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

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

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Instructor: Prof. P. P. Das, IIT Kharagpur. Date: Jun-Apr, 2018

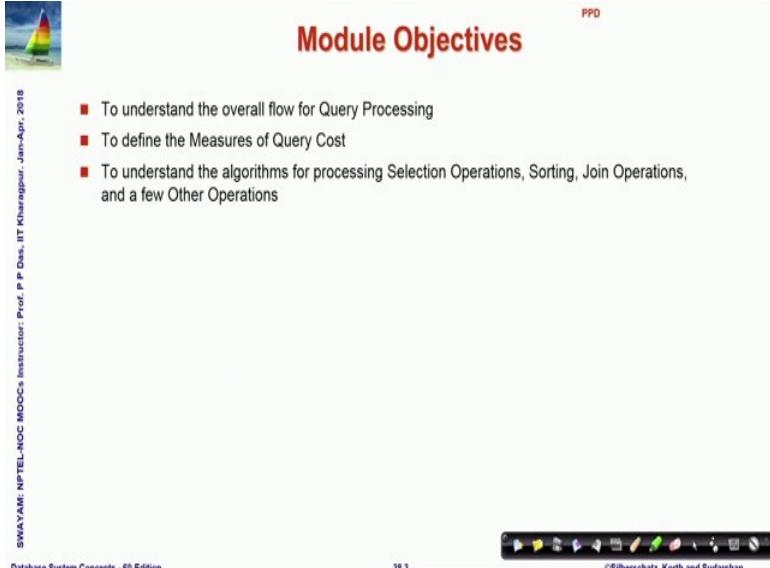
Database System Concepts - 8th Edition

38.2

©Silberschatz, Korth and Sudarshan

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

(Refer Slide Time: 00:36)



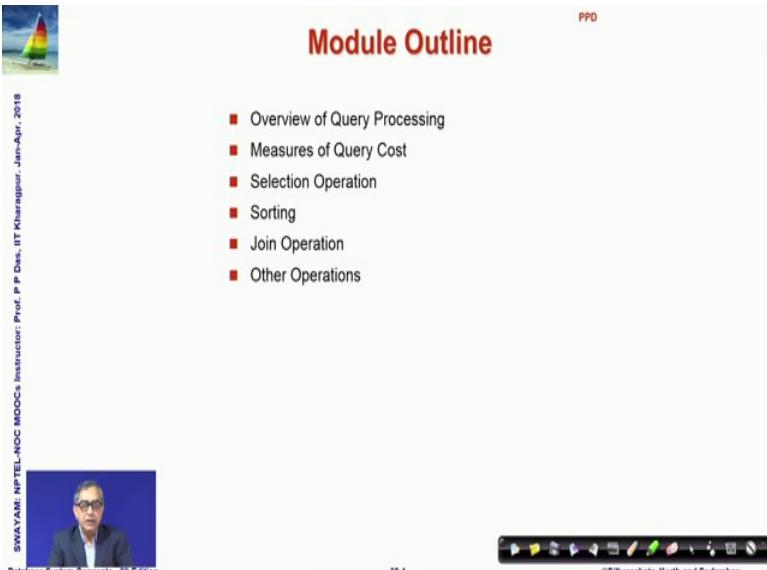
The slide title is "Module Objectives". It features a small sailboat icon in the top left corner and the acronym "PPD" in the top right corner. The main content is a bulleted list of objectives:

- 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

On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 6th Edition" and "38.3". The footer includes a navigation bar and the copyright notice "©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)



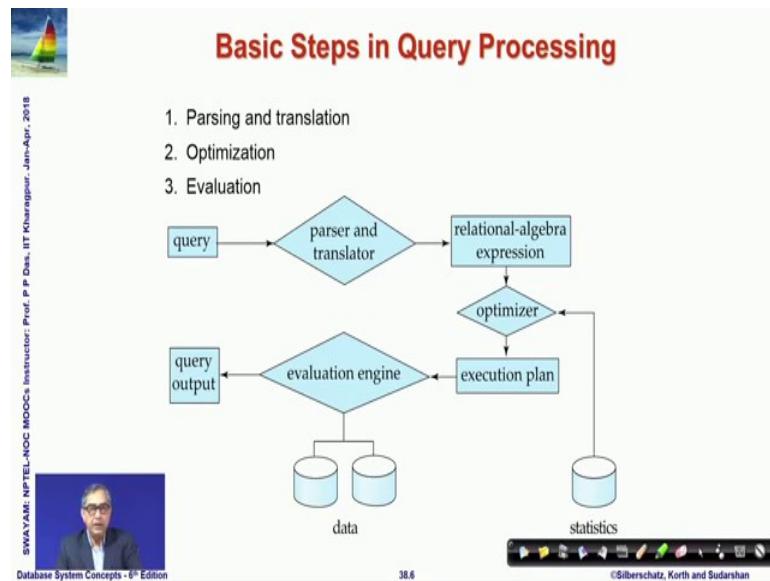
The slide title is "Module Outline". It features a small sailboat icon in the top left corner and the acronym "PPD" in the top right corner. The main content is a bulleted list of topics:

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

On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jan-Apr., 2018". In the bottom left corner, there is a video thumbnail showing a person's face. At the bottom, it says "Database System Concepts - 6th Edition" and "38.4". The footer includes a navigation bar and the copyright notice "©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 + 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 MOOCs Instructor: Prof. P P Dass, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition

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 $\text{salary} < 75000$,
 - or can perform complete relation scan and discard instructors with $\text{salary} \geq 75000$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Dass, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition

Now, if we take in terms of say a query a where σ 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 < 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 two 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)

The slide has a header 'Basic Steps: Optimization (Cont.)' with a sailboat icon. The content is organized into sections:

- **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 MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 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)

The slide has a header 'Measures of Query Cost' with a sailboat icon. The content lists factors contributing to 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. Desai, IIT Kanpur - 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 ma 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 write was successful.

(Refer Slide Time: 09:53)

The slide has a title 'Measures of Query Cost (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- 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

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner shows the copyright information: ©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)

The slide has a title 'Measures of Query Cost (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains two bulleted lists under red square bullet points:

- 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

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner includes the text 'SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018' and 'Database System Concepts - 8th Edition'. The bottom center says '38.13'. The bottom right has a copyright notice '©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)

The slide has a header 'Selection Operation' with a sailboat icon. It contains a list of bullet points under the heading 'File scan'. A handwritten note 'b_r * lrt + ts' is written next to the cost calculation. The footer includes a photo of a professor and the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur'.

- 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 consecutively
 - ▶ when there is an index available, binary search requires more seeks than index search

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, $(b_r / 2) * \text{block transfers} + 1 \text{ seek}$ so, if I if I write it in the notation that we had used earlier this $(b_r / 2) * \text{block transfer cost} + 1 \text{ seek cost}$.

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)

The slide has a title 'Selections Using Indices' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of algorithms and their details:

- **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
 - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right shows a standard Windows taskbar with icons for Start, Task View, File Explorer, Edge, Mail, Photos, and others.

Now, if now let us assume that it is the situation is such that we have some index on the B tree B + 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+ 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+ tree 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 $t_T + t_S$ 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)

The slide has a header 'Selections Using Indices' with a sailboat icon. The main content is a bullet-point list under '■ 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!

On the left margin, vertical text reads: SWAYAM, NPTEL-NOC, MDOC-4, Instructor: Prof. P. P. Desai, IIT Kanpur, Jan-Apr. 2018. At the bottom, there's a small video thumbnail of a man, the text 'Database System Concepts - 8th Edition', slide number '38.17', and copyright information '©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 two 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, $(h_i + 1) * (t_T + t_S)$ 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 = $h_i * (t_f + t_s) + b * t_r$
 - For $\sigma_{A \leq v}(r)$ just scan relation sequentially till first tuple $> v$; do not use index
 - Cost = $t_s + b * t_r$

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

Database System Concepts - 6th Edition

38.18

©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)

The slide has a title 'Selections Involving Comparisons' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains two bullet points:

- 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 = $(h_i + n) * (t_f + t_S)$
 - 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

At the bottom left, there is a small video thumbnail showing a man speaking. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8th Edition', '38.19', and '©Silberschatz, Korth and Sudarshan'.

But if we are doing a similar operation based 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 same block. So, they may be all distributed across different blocks and in such cases it may turn out that actually 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. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8th 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 Θ_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)

The slide has a header 'Algorithms for Complex Selections' with a sailboat icon. It lists several selection algorithms:

- **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. Deshpande, IIT Kanpur Date: Jan-Apr. 2018

Database System Concepts - 8th 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 Θ_1 will be there in the result, those which satisfy Θ_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 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)

The slide is titled "Sorting" in red. It contains the following bullet points:

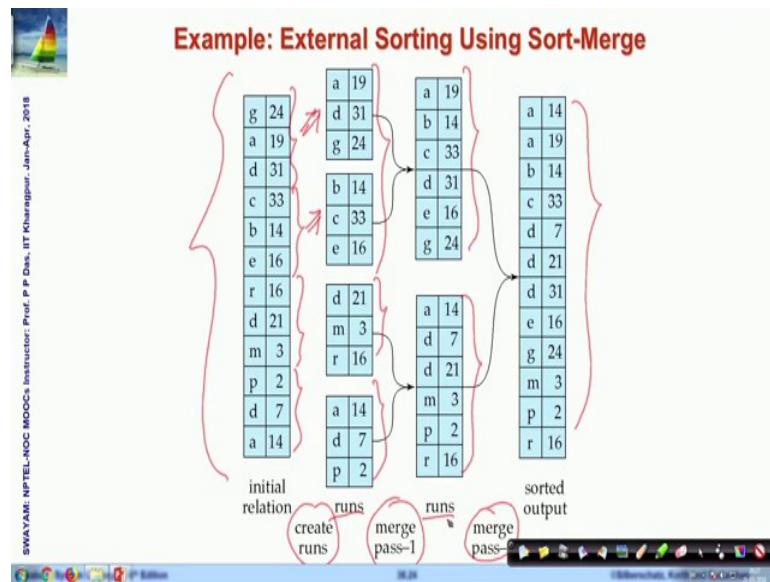
- We may build an index on the relation, and then use the index to read the relation in sorted order
 - May lead to one disk block access for each tuple
- For relations that fit in memory, techniques like quicksort can be used
- For relations that do not fit in memory, **external sort-merge** is a good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

On the left side of the slide, there is a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr. 2018". At the bottom left, it says "Database System Concepts - 8th Edition". On the right side, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number "38.23" is at the bottom center, and the copyright notice "©Silberschatz, Korth and Sudarshan" is at the bottom right.

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 + 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 fit 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 three just for illustration and make them and sort them in memory. So, take them so, take that many records which you can fit into the memory and sort them.

So, once you have sorted them then you can these are two 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 two 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)

The slide has a header 'External Sort-Merge' with a sailboat icon. The text describes the algorithm: 'Let M denote memory size (in pages)'. It lists two steps: 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).... The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8th Edition', '38.25', and '©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)

The slide continues the algorithm: 2. Merge the runs (N-way merge). We assume (for now) that $N < M$. It lists three steps: 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: The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8th Edition', '38.26', and '©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.)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- 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 - 8th Edition 38.27 ©Silberschatz, Korth and Sudarshan

And there are two 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

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- 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 - 8th 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 three strategies.

(Refer Slide Time: 27:15)

The slide has a title 'Nested-Loop Join' in red at the top right. On the left is a small image of a sailboat on water. The main content area contains a pseudocode algorithm for a theta 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  $\theta$ 
    if they do, add  $t_r \cdot t_s$  to the result.
  end
end
```

Below the pseudocode is a bulleted list of characteristics:

- 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

At the bottom left is a small video thumbnail showing a person speaking. The bottom right shows a navigation bar with icons and the text '38.30' and '©Silberschatz, Korth and Sudarshan'.

So, nested loop join what we are trying to do is very simple we have two relations we have two relations here r and s and we have a condition Θ and we are doing a Θ join. So, what needs to be done in terms of Θ join in the relational algebra what do we do we do a Cartesian product and then in the Cartesian product we check out this Θ 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 in s take the t_r, t_s pair and if they satisfy the condition Θ 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)

The slide has a header 'Nested-Loop Join (Cont.)' with a sailboat icon. The content includes a list of points about nested-loop joins:

- 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 MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr., 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 $b_r + b_l$ 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

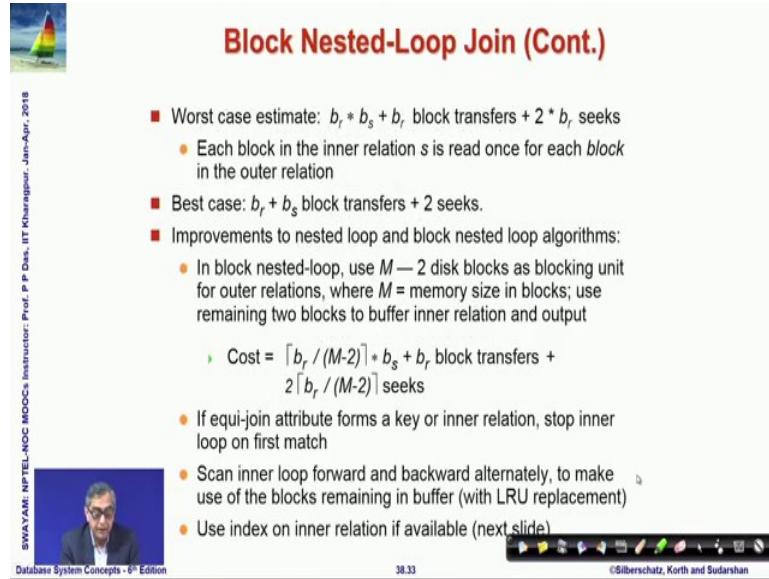
■ Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation

```
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
```

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

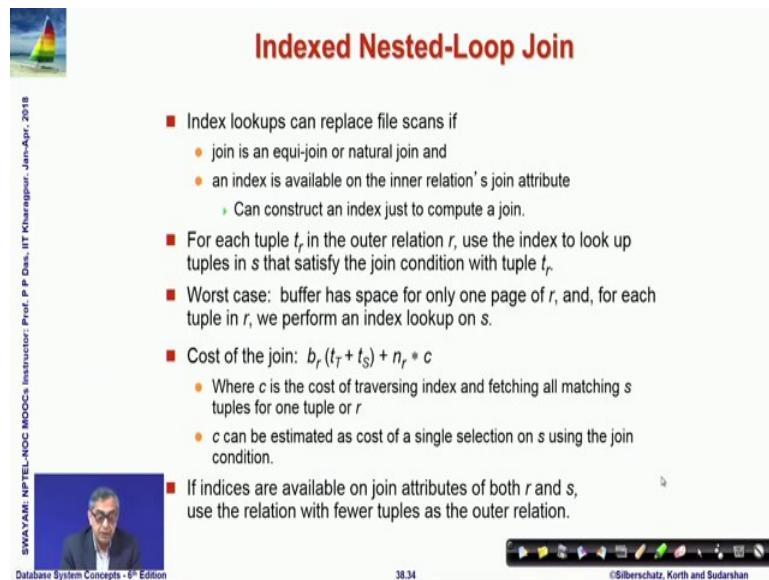
■ Improvements to nested loop and block nested loop algorithms:

- 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 MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018
Database System Concepts - 8th 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 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 of 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 MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018
Database System Concepts - 8th 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 two attributes between these two 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 MOOCs Instructor: Prof. P P Date, IIT Kanpur - Jan-Apr- 2018

Database System Concepts - 8th Edition 38.35 ©Silberschatz, Korth and Sudarshan

So, here is an example with the same students and takes example. So, it 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, 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 screenshot shows a presentation slide with the following elements:

- Section Title:** Other Operations
- List:** A bulleted list of six operations:
 - Duplicate elimination
 - Projection
 - Aggregation
 - Set Operations
 - Outer Join
- Navigation:** A small video player interface at the bottom left shows a video thumbnail of a man speaking, the title "Database System Concepts - 6th Edition", the time "38.37", and a copyright notice "©Silberschatz, Korth and Sudarshan".
- Decor:** A small sailboat icon is located in the top left corner of the slide area.

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 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 three we will quickly outline.

(Refer Slide Time: 35:05)

The slide has a header 'Other Operations' with a sailboat icon. The content includes a list of 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

At the bottom, there is a video player showing a speaker, the title 'Database System Concepts - 6th Edition', the time '38.38', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

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 side 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)

The slide has a header 'Other Operations : Aggregation' with a sailboat icon. The content includes a list of aggregation operations:

- **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

At the bottom, there is a video player showing a speaker, the title 'Database System Concepts - 6th Edition', the time '38.39', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

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-NOCO MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jam-Apr-2018

Database System Concepts - 8th Edition

38.40 ©Silberschatz, Korth and Sudarshan

So, we have in this module we have just given a very brief outline of 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.

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

Lecture - 39
Query Processing and Optimization/2: Optimization

Welcome to module 39 of database management systems, we have been discussing about query processing and optimization, in the last module.

(Refer Slide Time: 00:25)

The slide has a header "Module Recap" in red. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018". The main content area lists the following topics:

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

At the bottom, there is footer text: "Database System Concepts - 8th Edition", "39.2", and "©Silberschatz, Korth and Sudarshan". There is also a decorative toolbar icon at the bottom right.

We talked about the basic issues of query processing, what is the overview and the measures of query cost that would be used in terms of disk seek time and disk access time the read write time and we agreed we assume that we will ignore the CPU and other time for the time being. And then we took a look into how the certain SQL queries like the selection, the sorting which is required for other operations, different join operations and other aggregation and those kind of operations can be processed in a structured way.

(Refer Slide Time: 01:10)

Module Objectives

- To understand the basic issues for optimizing queries
- To understand how transformation of Relational Expressions can create alternates for optimization

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur, Jan-Apr. 2018

Database System Concepts - 8th Edition

39.3

©Silberschatz, Korth and Sudarshan

In view of this in this module we would like to understand the basic issues of optimizing queries. So, that the same query as we have seen little bit can be performed executed in multiple ways and we would like to choose the one which is the least cost possibly estimated cost should be the least.

So, we would need to understand two aspects, one is given a query how do I generate alternate queries that is in terms of the relational expression how a particular expression can be transformed into equivalent expressions and then we choose from these equivalence expressions for optimization. So, these are the two topics to discuss.

(Refer Slide Time: 01:57)

Introduction

- Alternative ways of evaluating a given query
 - Equivalent expressions
 - Different algorithms for each operation

$\text{course}(\underline{\text{course id}}, \text{title}, \text{dept name}, \text{credits})$
 $\text{instructor}(\underline{\text{ID}}, \text{name}, \text{dept name}, \text{salary})$
 $\text{teaches}(\underline{\text{ID}}, \underline{\text{course id}}, \underline{\text{sec id}}, \text{semester}, \text{year})$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur

Database System Concepts - 8th Edition

39.6

©Silberschatz, Korth and Sudarshan

So, to introduce on the query optimization let us take a simple example. So, I am referring to the university database that we have discussed earlier, it has three relations as listed above and using that we want to do the perform the query where we join **instructor** \bowtie **teaches** \bowtie **course** and do a σ on that based on department name. So, this will give us naturally the instructors who are in teaching some course in the instructor is in music department and is teaching some course there and we want the name and title of these. So, we want the name of the instruction instructor and the title of the course that the person is teaching.

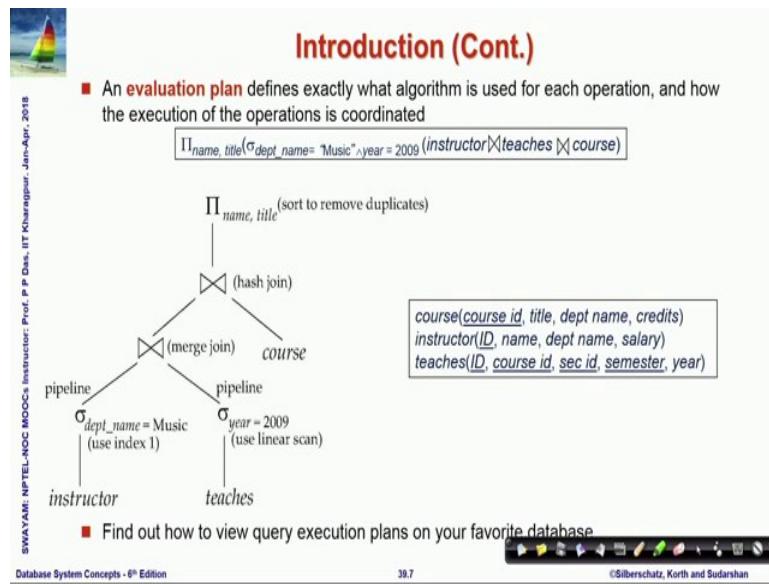
So, if we write the query in equivalent relational form this is what we will get to see and this is basically is a first join happening here then the next join happening, find next the σ and finally, the $\Pi_{\text{name}, \text{title}}$ happening here which will give us the result of this query. Now what we observe is it is possible that if you look at carefully the department name should be music.

So, if we look into these relations then we can easily figure out that instructor has the department name attribute. So, in this after this joint if a tuple has to qualify through this selection then the instructors in the instructor relation the department name of that join people must be music otherwise it will not get selected.

So, what if instead of doing the join and then doing the selection as we are doing here if we first do a selection on the instructor relation itself and then join it with the join of

teachers and courses and finally, do the projection we should actually get the same result. So, these are what is it is a simple example of what are equivalent relational expressions that we can make use of in terms of our query processing.

(Refer Slide Time: 04:20)



So, what given that this is another example we were showing. So, here the query is to find the teacher and instructor and the course name for back the instructor is from department of the music and the course he taught is in the year 2009. So, we want these and corresponding to the equivalent at SQL if we write the relational expression then the in relational algebra this is what it looks like. So, what we can eventually observe from this that again as we observed last time department name is an is an instructor.

So, if we are doing a final $\sigma_{dept_name = music}$ then it is possible that I can first filter the instructor relation with the department name being music that should not affect the result. At the same time $\sigma_{year=2009}$, which happens only in the teacher's relation.

So, instead of actually filtering it after the join $(\sigma_{dept_name = music} \bowtie \sigma_{year=2009}) \bowtie course$ do $\Pi_{name, title} ((\sigma_{dept_name = music} \bowtie \sigma_{year=2009}) \bowtie course)$. So, here what we show that along with this tree the parse tree of the query in relational algebra we are also trying to put some annotations to say how this particular query will be processed.

So, if we want to find out the tuples where the year is 2009 and there is no specific index on that or anything to for doing this selection as we have seen earlier the best way would

be to use a linear scan. Whereas, if I am trying to do a selection with department name is equal to is music there will be a could be an index one the secondary index based on the department name. So, I will be able to use that index to find this.

So, these when we are doing this putting and that here we will use this index, here we will use linear scan these are what is called annotations which tell the query processing engine that how the query should actually be executed. Then they will be pipelined in the sense that this will be put one after the other actually these two can happen in parallel and then we will use a merge join to join these two then this merge that is joined result would be joined with course based on now the course is already indexed in course id and this joined relation of instructor and teachers will have id and course id on which they will have index.

So, we can use a hash join on that we did not discuss about merge join, hash join as processing steps in detail, but right now just assume that these are different ways of doing join which can make things efficient and these are the annotations which are generated in the process. And such a query tree such a query parse tree with the annotation is called an execution plan so, that or the evaluation plan which the query processing engine would be able to use to actually execute and find the results.

(Refer Slide Time: 07:58)

The slide has a header 'Introduction (Cont.)' with a sailboat icon. The main content is a bulleted list:

- Cost difference between evaluation plans for a query can be enormous
 - E.g. seconds vs. days in some cases
- Steps in **cost-based query optimization**
 1. Generate logically equivalent expressions using **equivalence rules**
 2. Annotate resultant expressions to get alternative query plans
 3. Choose the cheapest plan based on **estimated cost**
- Estimation of plan cost based on:
 - Statistical information about relations.
 - ▶ Examples: number of tuples, number of distinct values for an attribute
 - Statistics estimation for intermediate results
 - ▶ to compute cost of complex expressions
 - Cost formulae for algorithms, computed using statistics

Navigation icons are at the bottom right, and footer text includes 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6th Edition', '39.8', and '©Silberschatz, Korth and Sudarshan'.

So, this is the basic approach so, for optimization what we need to do we need to find out that particular evaluation plan, that particular order of we the evaluation and the use of

algorithms, the use of indexes which will make the query possibly most efficient. And that cost difference could be really really huge in a real database it could vary between seconds in one way of doing the query or in terms of number of days if we do it in a non optimal way in a non optimized way.

So, typical steps in this kind of query based optimization would be to first generate the candidates I am sorry first generate the candidates that is generate the equivalent expressions that is given a query I have one relational expression and we would like to generate equivalent expression using a set of equivalence rules we will see what these equivalence rules are.

So, that this equivalent queries expressions can any one of them can be actually executed and we then annotate them to with the result the resultant expression we annotate to get different query plans evaluation plans. And then we put the cost estimates based on the cost structure that say we had used in the other in the earlier module and from these alternate evaluation plans we will choose the one that will have the least estimated cost.

So, the cost can be based on as we had seen earlier it could be based on number of tuples, number of distinct values frequently used attributes and so, on and we will use statistics for also intermediate results that if there are intermediate results to be stored we will make estimates of what would be the size of that result because it needs to fit into memory for optimal execution, the cost formula for different algorithms will also be used through statistics.

So, based on all these estimation which will have an estimated cost and based on that we will choose the particular evaluation plan which looks to be the best and that is the crux of the query optimization strategy. So, as we have seen the first step is to be able to generate alternate expressions equivalent expressions through transformations. So, we look through the relational algebra operators again.

(Refer Slide Time: 10:27)

Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
 - Note: order of tuples is irrelevant
 - We do not care if they generate different results on databases that violate integrity constraints
- In SQL, inputs and outputs are multisets of tuples
 - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance.
- An **equivalence rule** says that expressions of two forms are equivalent
 - Can replace expression of first form by second, or vice versa

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

39.10

©Silberschatz, Korth and Sudarshan

And check what are what is meant by equivalence of two relational expressions. So, two relational expressions are equivalent if they generate the same set of tuples for any instance of the database, it is not enough to just show that for one instance it gives the same result.

So, two expressions are equivalent for in if I take any legal instance of the database then it must be equivalent and in this process we can note that the order of tuples are really relevant that we have told repeatedly and also we would make sure that the results are same provided the database provided the relation satisfy all the integrity constraints. If they violate integrity constraints then it is a problem of the user then we the database really does not care if the two expressions will give equivalent or equal results. We also note that in SQL input output could be multisets so, the same thing has to be satisfied in terms of multisets. So, based on this we define an equivalence a set of equivalence rule that say that two expressions are equivalent so, you can use that rule to transform one expression by the other and vice versa.

(Refer Slide Time: 11:47)

The slide has a header 'Equivalence Rules' in red. On the left, there is a small sailboat icon and some vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018'. Below the header is a list of four equivalence rules:

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections
$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$
2. Selection operations are commutative
$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$
3. Only the last in a sequence of projection operations is needed, the others can be omitted
$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$
4. Selections can be combined with Cartesian products and theta joins
 - a. $\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$
 - b. $\sigma_{\theta_1}(\sigma_{\theta_2}(E_1 \times E_2)) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

At the bottom, there is a video player showing a person speaking, and a navigation bar with icons.

So, let us take a look into this expression so, most of these expressions are relatively easy to understand. They can be formally proved using the corresponding set theoretic condition of the relational expressions. So, for every relational expression we had a set theoretic condition that we had studied so, using those you can prove that we will not do the proof of these equivalence relations here, but it you should be able to do that very easily.

So, he said if we have a conjunctive selection of a relation based on two conditions Θ_1 and Θ_2 , then we should be able to apply the selection first based Θ_2 , σ_{Θ_2} and then based on Θ_1 , $\sigma_{\Theta_1}(\sigma_{\Theta_2})$ or actually $\sigma_{\Theta_2}(\sigma_{\Theta_1})$ vice versa because conjunction is commutative so, the selection operation is also commutative. So, this gives us two transformation rules and we might want to use any so, these all will give equivalent form.

So, applying these two rules we get three relational expressions which are all equivalent this, this and this are all equivalent, but if you actually think in terms of processing the query and the cost involved you will be able to figure out that naturally the cost of doing this may be relatively more involved because you have the relation and then on the whole relation you are applying the conditions, but here if you do for example, if you first do say first apply Θ_2 certainly by that selection the relation would become much smaller. So, applying Θ_1 on that would be easier or vice versa depending on whichever is

a smaller set there could be other for example, if you have a sequence of projections then naturally in that sequence the last projection that you have done is what is retained.

So, if we have a sequence of projections that is simply doing the last projection all other projections can actually be ignored. The selection can be combined with Cartesian product and Θ join also that is taking a Cartesian product and then doing a selection is equivalent to doing a Θ join this of course, is almost the definition. And if I have a join $\sigma_{\Theta_1}(E_1 \bowtie_{\Theta_2} E_2)$ it is equivalent to doing a $E_1 \bowtie_{\Theta_1 \wedge \Theta_2} E_2$

So, these are all equivalent forms so, these are equivalent in the sense that left hand side and right hand side are interchangeable. So, it does not mean that the left hand side implies the right hand side or vice versa it means that either of them implies the other, they are really equivalent in that sense.

(Refer Slide Time: 14:33)

Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative
 $E_1 \bowtie_{\Theta} E_2 = E_2 \bowtie_{\Theta} E_1$

6. (a) Natural join operations are associative:
 $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

(b) Theta joins are associative in the following manner:
 $(E_1 \bowtie_{\Theta_1} E_2) \bowtie_{\Theta_2 \wedge \Theta_3} E_3 = E_1 \bowtie_{\Theta_1 \wedge \Theta_3} (E_2 \bowtie_{\Theta_2} E_3)$
where Θ_2 involves attributes from only E_2 and E_3 .

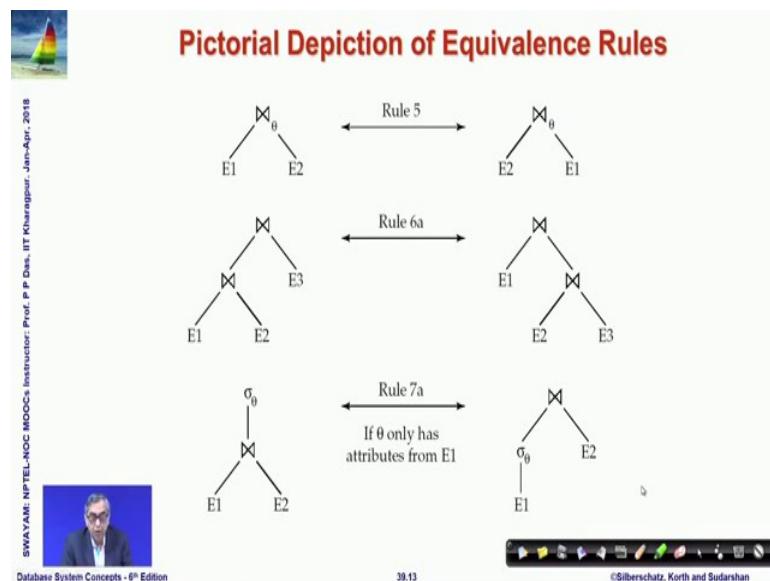
SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Date, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
39.12
©Silberschatz, Korth and Sudarshan

Then Θ join operation are natural or natural join operations are commutative, we can change interchange their relations.

So, this might impact for example, you have seen the algorithms of nested join and block join block nested join algorithms; obviously, depending on the size of the relations the cost of doing $E_1 \bowtie_{\Theta} E_2$ or $E_1 \bowtie E_2$ and $E_2 \bowtie_{\Theta} E_1$ may be different and we would choose the one which has a lesser cost.

Next set of transformation rules tell us that the join operation is associative which is obvious Θ joins are associative in the in this specific manner also, that is if I do a $(E_1 \bowtie_{\Theta_1} E_2) \bowtie_{\Theta_2} E_3$ then we may be able to take out the condition Θ_3 outside provided Θ_2 the condition Θ_2 uses only the attributes of Θ , E_1 and E_2 so, $E_1 \bowtie_{\Theta_1} E_2 \bowtie_{\Theta_2} E_3$ it should be possible to do that. Look here that on the left hand side that restriction is not there on the condition Θ_2 it could also use attributes of E_1 , but if it does not then it is possible to simplify it in this manner and these are the equivalent rules that we have.

(Refer Slide Time: 16:01)



So, often we will draw the rules in this form so, just to explain you one so, this shows the associativity of join.

So, here what you are saying is first you do E_1, E_2 and with the result you join E_3 , $(E_1 \bowtie_{\Theta} E_2) \bowtie_{\Theta_3} E_3$ here you are saying first you do E_2, E_3 and then you join with E_1 . $(E_2 \bowtie_{\Theta} E_3) \bowtie_{\Theta_1} E_1$ So, this is basically associativity this basically is commutativity of Θ join. So, we will often draw them in such forms of parse trees and show the equivalence that becomes easy to understand for example, in this case you are doing a join and then doing the selection and here you are doing it select early you are doing a select early on this relation E_1 of course, you will be able to do this provided Θ contains only attribute from E_1 , if Θ contains attributes from both E_1 and E_2 this transformation will not be applicable this transformation will not be possible.

(Refer Slide Time: 17:00)

The slide has a header 'Equivalence Rules (Cont.)' in red. On the left, there is a small image of a sailboat and some text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. In the center, there is a list of rules. Rule 7 states: 'The selection operation distributes over the theta join operation under the following two conditions: (a) When all the attributes in Θ_0 involve only the attributes of one of the expressions (E_1) being joined' followed by the equation $\sigma_{\Theta_0}(E_1 \bowtie_{\Theta} E_2) = (\sigma_{\Theta_0}(E_1)) \bowtie_{\Theta} E_2$. Rule 7(b) states: '(b) When Θ_1 involves only the attributes of E_1 and Θ_2 involves only the attributes of E_2 ' followed by the equation $\sigma_{\Theta_1 \wedge \Theta_2}(E_1 \bowtie_{\Theta} E_2) = (\sigma_{\Theta_1}(E_1)) \bowtie_{\Theta} (\sigma_{\Theta_2}(E_2))$. At the bottom, there is a video thumbnail showing a man, the text 'Database System Concepts - 8th Edition', the page number '39.14', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, these are some more of the transformation rules the selection operation distributes over Θ join operation. So, I can you can I can see here that there is a Θ join and then I am doing a selection. So, I can first do the selection and then do the Θ join of course, for the selection it must involve only the attributes of E_1 , otherwise this will not be valid.

And similarly, another distribution rule is shown here where you are doing a selection on conjunction on $\sigma_{\Theta_1 \wedge \Theta_2}(E_1 \bowtie_{\Theta} E_2)$ then you should be able to actually distribute based on the condition Θ_1 and Θ_2 , if Θ_1 involves only the attributes of E_1 and Θ_2 involves only the attributes of E_2 , $\sigma_{\Theta_1}(E_1) \bowtie_{\Theta} \sigma_{\Theta_2}(E_2)$. These are these are pretty straightforward rules if you think about the corresponding set theoretic reason.

(Refer Slide Time: 17:58)

The slide has a header 'Equivalence Rules (Cont.)' in red. On the left, there is a small image of a sailboat on water. The main content is as follows:

8. The projection operation distributes over the theta join operation as follows:

(a) if Θ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\Theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\Theta (\Pi_{L_2}(E_2))$$

(b) Consider a join $E_1 \bowtie_\Theta E_2$.

- Let L_1 and L_2 be sets of attributes from E_1 and E_2 , respectively
- Let L_3 be attributes of E_1 that are involved in join condition Θ , but are not in $L_1 \cup L_2$, and
- Let L_4 be attributes of E_2 that are involved in join condition Θ , but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\Theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\Theta (\Pi_{L_2 \cup L_4}(E_2)))$$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8th Edition

39.15

©Silberschatz, Korth and Sudarshan

So, the other rules will be in terms of projection so, you can have if you have union projection like this then you would be able to break it down over to do separate projections and their Θ join and in case you have a Θ join of E_1, E_2 based on Θ and if L_1 and L_2 are the attributes of these two relations.

So, if L_3 be the attributes of E_1 that are involved in the join condition Θ and L_4 are what are. So, you have the join condition Θ . So, what you are saying that the attributes L_3 of you are not involved here and attributes L_4 of E_2 are involved here, but they are not so, in terms of $L_1 \cup L_2$ so, then this kind of I should be able to distribute the projection in steps and make the results smaller.

(Refer Slide Time: 19:04)



Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

■ (set difference is not commutative).

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. The selection operation distributes over \cup , \cap and $-$.

$$\sigma_0(E_1 - E_2) = \sigma_0(E_1) - \sigma_0(E_2)$$

and similarly for \cup and \cap in place of $-$

Also: $\sigma_0(E_1 - E_2) = \sigma_0(E_1) - E_2$
and similarly for \cap in place of $-$, but not for \cup

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$


SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
39.16 ©Silberschatz, Korth and Sudarshan

So, this is another possible transformation that now finally, the set theoretic operations have their normal set rules so, union and intersection are commutative, naturally set difference is not commutative, union intersection are associative as well. The selection operation distributes over union, intersection and set difference and the projection also distributes over union. So, you can see the exceptions here you can reason that out.

(Refer Slide Time: 19:35)



Exercise

■ Create equivalence rules involving

- The group by/aggregation operation
- Left outer join operation



SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
39.17 ©Silberschatz, Korth and Sudarshan

So, we have in short we have presented a set of different transformation rules by each which you can make equivalent expressions and between two equivalent expressions or

more equivalent expressions our objective will always be to choose the one whose evaluation plan will have a lesser cost. So, as an exercise I have left that you can create equivalence rules that involve group by or aggregation operations or different kinds of outer join, left outer join, right outer join and so, on so, please work those out.

(Refer Slide Time: 20:05)

Transformation Example: Pushing Selections

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
 - $\Pi_{name, title}(\sigma_{dept_name = "Music"}(instructor) \bowtie (\text{teaches} \bowtie \Pi_{course_id, title}(course)))$
- Transformation using rule 7a
 - $\Pi_{name, title}((\sigma_{dept_name = "Music"}(instructor)) \bowtie (\text{teaches} \bowtie \Pi_{course_id, title}(course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined

Diagram illustrating Rule 7a:

```

    graph LR
      E1 --> E2
      E1 --> S
      S --> E2
      S --> O
      O --> E2
      style S fill:none,stroke:none
      style O fill:none,stroke:none
  
```

The diagram shows two relations, E1 and E2, connected by a join symbol. A third node, S, is connected to both E1 and E2. A curved arrow labeled "Rule 7a" points from the original query structure to this diagram, indicating that applying the rule results in a smaller intermediate relation S.

Let us move on to a couple of examples so, here is a query here the relations from the university database, here is a query find the names of all instructors in the music department along with the titles of the course they teach. So, a while ago we saw this so, this is what the query is in the relational algebra form and if you use the transformation rule of select early the rule 7a, whose transformation I have just shown here then you should be able to hear what you are doing is you are first doing a join of teaches and the projection of course. And then joining instructor with that and finally, doing the selection, but you should you would be able to do this select early because it involves the attribute department name which is the attribute of instructor alone here.

So, you should be able to first do this selection, mind you here courses also show that there is an attribute department name, but actually the courses is being used after projection. So, after projection in the projected relation there is no department name. So, department name is involved only the instructor.

So, I can first I can take this do early, I can do this selection early and take this out and then do the final join operation. So, in that process possibly the this will reduce the size

of the relation to be joined significantly because you do not naturally expect to be to have too many instructors in the music department. So, the instructor relation after the selection would become much smaller.

(Refer Slide Time: 21:52)

Example with Multiple Transformations

■ Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught

- $\Pi_{name, title}(\sigma_{dept_name = "Music"} \wedge year = 2009 (instructor \bowtie (teaches \bowtie \Pi_{course_id, title}(course))))$

■ Transformation using join associatively (Rule 6a):

- $\Pi_{name, title}(\sigma_{dept_name = "Music"} \wedge year = 2009 ((instructor \bowtie teaches) \bowtie \Pi_{course_id, title}(course)))$

■ Second form provides an opportunity to apply the “perform selections early” rule, resulting in the subexpression

$$\sigma_{dept_name = "Music"}(instructor) \bowtie \sigma_{year = 2009}(teaches)$$

course(course_id, title, dept_name, credits)
 instructor(ID, name, dept_name, salary)
 teaches(ID, course_id, sec_id, semester, year)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018
 Database System Concepts - 8th Edition
 39.19
 ©Silberschatz, Korth and Sudarshan

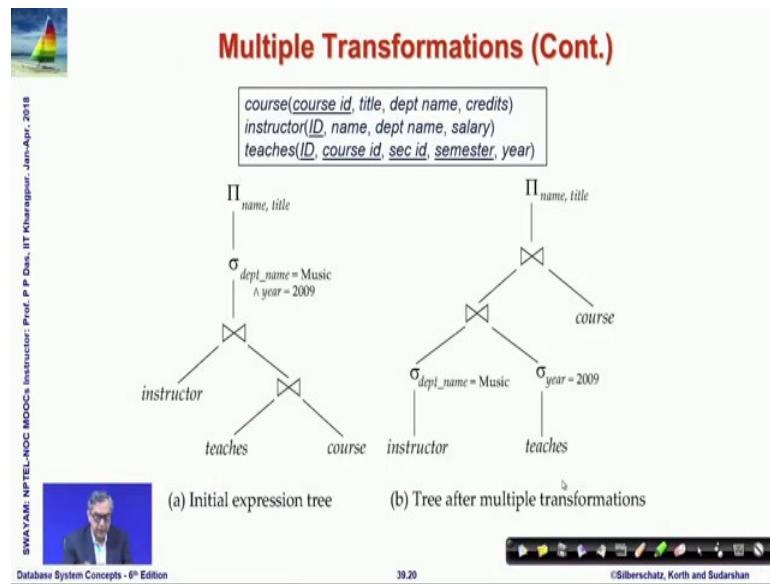
So, this is one example, this is another example query we are taking find the names of all instructors in the music department, we have taught a course in 2009 along with the titles of the course they taught. So, here is a the corresponding here is the corresponding relational query which you can convince yourself is indeed the same.

And then we can transform using the rule 6a which I have shown here which is basically the associativity of joint because there are two join relations and we are using the associativity of join to first join the here then second and third relations are joined first and then the first relation is joined with that. Here we are using associativity of joining the first and second and then using the third, now if you join first and second and then it is possible that the department name actually the department name is an instructor and the other condition is here which is in the teaches.

So, this department name is in the instruction here is in the teaches. So, I should be able to do select early, I should be able to select on the instructor based only on the department name being music and also select early on the teachers by using a year as 2009 and then do their joint.

So, naturally each one of these will become much smaller corresponding to the whole instructor or teaches relation therefore, their join will also be smaller. So, which means eventually this part this part of the query will become much smaller in size in the result and the consequent second join would be much smaller to perform. So, this will naturally give it whereas, if we had done all of these join earlier we would have used the whole of the teaches and the instructor relations and that would have been quite a lot of tuples would have been there.

(Refer Slide Time: 23:57)



So, these are different example so, this is the same example being shown in terms of the tree parts tree structure so, we can convince yourself by using the transformation rules that they are actually equivalent.

(Refer Slide Time: 24:12)

Transformation Example: Pushing Projections

- Consider: $\Pi_{name, title}(\sigma_{dept_name = "Music"}(instructor) \bowtie \text{teaches}) \bowtie \Pi_{course_id, title}(course)$
- When we compute
 $(\sigma_{dept_name = "Music"}(instructor) \bowtie \text{teaches})$
we obtain a relation whose schema is:
 $(ID, name, dept_name, salary, course_id, sec_id, semester, year)$
- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:
$$\begin{aligned} &\Pi_{name, title}(\Pi_{name, course_id}(\sigma_{dept_name = "Music"}(instructor) \bowtie \text{teaches})) \bowtie \\ &\quad \Pi_{course_id, title}(course) \end{aligned}$$
- Performing the projection as early as possible reduces the size of the relation to be joined

$$\begin{aligned} &\text{course}(course_id, title, dept_name, credits) \\ &\text{instructor}(ID, name, dept_name, salary) \\ &\text{teaches}(ID, course_id, sec_id, semester, year) \end{aligned}$$

$$\begin{aligned} &\Pi_{L_1 \cup L_2}(E_1 \bowtie E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2)) \quad \text{8a} \\ &\Pi_{L_1 \cup L_2}(E_1 \bowtie E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_2}(E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_1}(E_2))) \quad \text{8b} \end{aligned}$$

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

And then certainly they give you a significant advantage and now this is an example which show that you can push the projection.

So, here is what you wanted to do and while we compute this we will get a relation of this form and we can push, we can use these rules rule 8a and 8b, this is rule 8a, this is rule 8b by this we can push the projections inside and make the relations smaller because now if you push the projection then you are actually cutting down on the on the number of columns. So, your relation becomes smaller that will make the with the projection this will reduce and when you project also there will be duplicates which will get removed so, in every way your relation becomes smaller in size and your subsequent join operations would be more efficient.

(Refer Slide Time: 25:02)

The slide features a sailboat icon in the top left corner. The title 'Join Ordering Example' is at the top right. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area has a red border. It contains two bullet points:

- For all relations r_1 , r_2 and r_3 ,
$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

(Join Associativity)
- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose
$$(r_1 \bowtie r_2) \bowtie r_3$$

so that we compute and store a smaller temporary relation

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', the time '39:22', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

You can also see these are examples where you can take care of the fact that you can reorder joining to get better result for example, the you can if I have to do join three relations like this, I could do either this first or this first. Now if I do this first ($r_1 \bowtie (r_2 \bowtie r_3)$) then this is a temporary relation which I will need to maintain in memory and then join with r_1 . If I do this first then this will be a temporary relation and then I will join it with $r_3 ((r_1 \bowtie r_2) \bowtie r_3)$. So, if this is large enough then compared to this then I will be better off by doing this and will be able to have a more optimized execution of the query. So, here is an example of that again two joints.

(Refer Slide Time: 25:46)

The slide features a sailboat icon in the top left corner. The title 'Join Ordering Example (Cont.)' is at the top right. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area has a red border. It contains several bullet points:

- Consider the expression
$$\Pi_{name, title}(\sigma_{dept_name = "Music"}(instructor) \bowtie teaches) \bowtie \Pi_{course_id, title}(course))$$
- Could compute $teaches \bowtie \Pi_{course_id, title}(course)$ first, and join result with
$$\sigma_{dept_name = "Music"}(instructor)$$
- but the result of the first join is likely to be a large relation
- Only a small fraction of the university's instructors are likely to be from the Music department
 - it is better to compute
$$\sigma_{dept_name = "Music"}(instructor) \bowtie teaches$$
 first

At the bottom, there is a box containing the following table definitions:

course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
teaches(ID, course_id, sec_id, semester, year)

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', the time '39:23', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, what we are trying to show is when we are having this instead of actually are actually trying to compute this join first because we expect that this set to be much smaller, if this set is much smaller then naturally this joint would be much smaller and it is more likely to fit into the memory then if we had just done teaches and course id's which I expected to be large relations.

(Refer Slide Time: 26:20)

The slide has a title 'Enumeration of Equivalent Expressions' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list:

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
 - Repeat
 - apply all applicable equivalence rules on every subexpression of every equivalent expression found so far
 - add newly generated expressions to the set of equivalent expressions
 - Until no new equivalent expressions are generated above
- The above approach is very expensive in space and time
 - Two approaches
 - Optimized plan generation based on transformation rules
 - Special case approach for queries with only selections, projections and joins

At the bottom left is a small video thumbnail showing a person speaking. The footer contains the text 'SWAYAM-NPTEL-NOCC Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8th Edition', '39.24', and '©Silberschatz, Korth and Sudarshan'.

So, basically therefore, the strategy turns out that given a query you will have to generate it whole lot of equivalent expressions. So, the number of equivalent expressions could be really really large. So, we will have to systematically generate all these alternate equivalent expressions and apply by applying these transformation rules that we have just seen and we will have to continue till no new expression can be generated and then we have to evaluate each one of them based on their evaluation plan.

So, this could be very expensive because the number of alternates could be really really large. So, the optimize plan generation is also based on the transformation rules and we may have only different special kits approaches to take care of the common optimization plans that we might have.

(Refer Slide Time: 27:16)

Implementing Transformation Based Optimization

■ Space requirements reduced by sharing common sub-expressions:

- when E1 is generated from E2 by an equivalence rule, usually only the top level of the two are different, subtrees below are the same and can be shared using pointers
 - E.g. when applying join commutativity

■ Same sub-expression may get generated multiple times

- Detect duplicate sub-expressions and share one copy

■ Time requirements are reduced by not generating all expressions

- Dynamic programming

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018
Database System Concepts - 8th Edition
39.25
©Silberschatz, Korth and Sudarshan

Now, one way to do that is for example, if we are looking at say the join of two relations E1 and E2 here then; obviously, if we do certain transformations with this join that does not change the way E1 and E2 are actually evaluated.

So, if that be the case then in terms of generating the alternate we do not really need to keep or evaluate all of the expressions the whole of the expression in every alternate. We could actually do something like sorry we could actually do something like we could have a join and then instead of really replicating the whole of the evaluation of E1 and evaluation of E2, we can simply make pointers to the same sub trees which are called basically sub expression optimization.

If you have studied the expression optimization in compiler at any point of time you will understand this very well, this is the same strategy which is used here. So, if you can detect duplicate sub expressions then you can have only one copy and you can make the things more efficient to run.

So, the general strategy for doing this is a dynamic programming we would not be able to cover that in the current course, but just know that all these explosion of generating alternate and choosing the best one is usually handled in terms of dynamic programming which is a strategy to make sure that if we have solved a sub earlier then I do not need to solve that sub problem again we can just reuse that same earlier result and go ahead with that.

So, by this way we can common sub expressions we may only find the plan for a best plan for a common sub expression only once and then use it subsequently again.

(Refer Slide Time: 29:24)

Module Summary

- Understood the basic issues for optimizing queries
- For every relational expression, usually there are a number of equivalent expressions that can be created by simple transformations
- Final execution plan can be created by choose the estimated least cost expression from the alternates

SWAYAM: NIFTEL-MOOCs Instructor: Prof. P P Das, IIT Kharagpur Jan-Apr - 2018

Database System Concepts - 8th Edition 39.26 ©Silberschatz, Korth and Sudarshan

So, in this module covered starting from the notions of query processing in the earlier module, we have discussed the basic issues of optimizing queries and we have shown that using a set of simple transformational rules you can convert a relational expression into a number of equivalent relational expressions. And then you can evaluate them based on the estimated cost that the model that you are using and choose the best one and dynamic programming is usually a good way of doing that.

So, in through the previous module and this one we have taken a very elementary look I should say, but this will give you some idea of how actually an SQL query is transformed into relational algebra parsed and translated into relational algebra and how equivalent expressions for that relational algebra expression is generated and evaluated.

And finally, an evaluation plan is made where specific choice is made for the different algorithms for doing different operations and that final plan which is which has the best cost is passed on to the query processing engine to query evaluation engine. And that then goes forward and does the B tree and disk operations in according to the plan and produces the best result in possibly the best shot is possible time period.

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

Lecture – 40
Course Summarization

Welcome to module 40 of Database Management Systems. This is for course summarization this is the last module of the course.

(Refer Slide Time: 00:23)

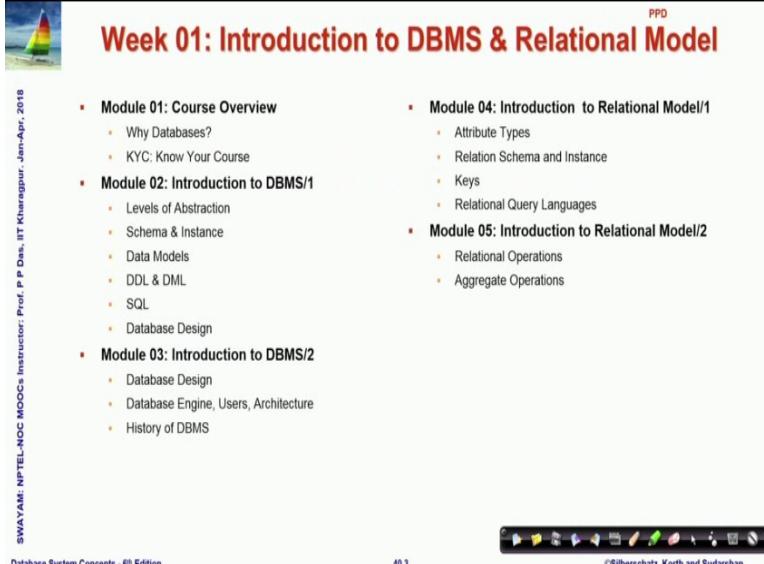
The slide is titled "Course Recap" in red text at the top right. In the top left corner, there is a small image of a sailboat on water. The footer contains several lines of text and icons:

- SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018
- Database System Concepts - 8th Edition
- 40.2
- ©Silberschatz, Korth and Sudarshan

At the bottom, there is a horizontal bar with various presentation control icons (e.g., back, forward, search).

So, I would just start with doing a quick recap of what all did we cover and what we expectedly learnt.

(Refer Slide Time: 00:32)



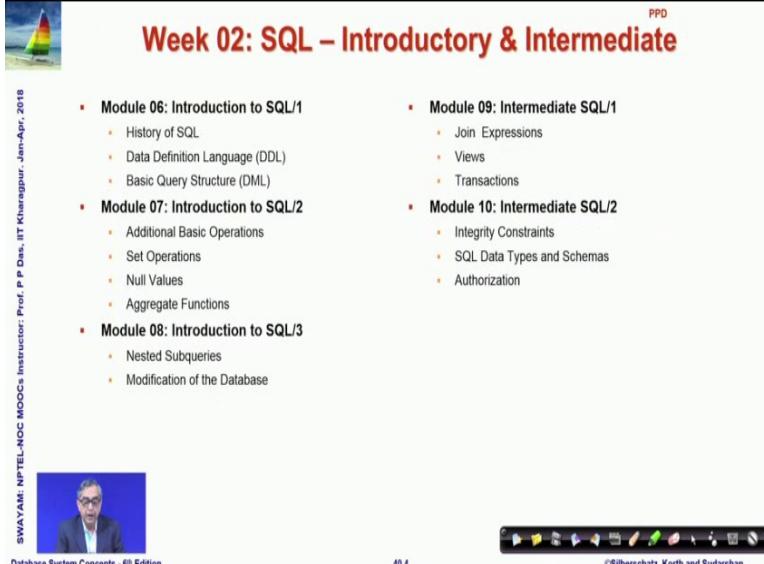
The slide title is "Week 01: Introduction to DBMS & Relational Model". It features a small sailboat icon in the top left corner and the letters "PPD" in the top right corner. The slide content is organized into two columns of bullet points:

Module	Topics
Module 01: Course Overview	Why Databases? KYC: Know Your Course
Module 02: Introduction to DBMS/1	Levels of Abstraction Schema & Instance Data Models DDL & DML SQL Database Design
Module 03: Introduction to DBMS/2	Database Design Database Engine, Users, Architecture History of DBMS
Module 04: Introduction to Relational Model/1	Attribute Types Relation Schema and Instance Keys Relational Query Languages
Module 05: Introduction to Relational Model/2	Relational Operations Aggregate Operations

On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 8th Edition" and "40.3". On the right, there is a decorative toolbar and the copyright notice "©Silberschatz, Korth and Sudarshan".

In the week 1, we talked primarily of Introduction to Database Management System and the Relational Model which is the foundation of a database system.

(Refer Slide Time: 00:42)



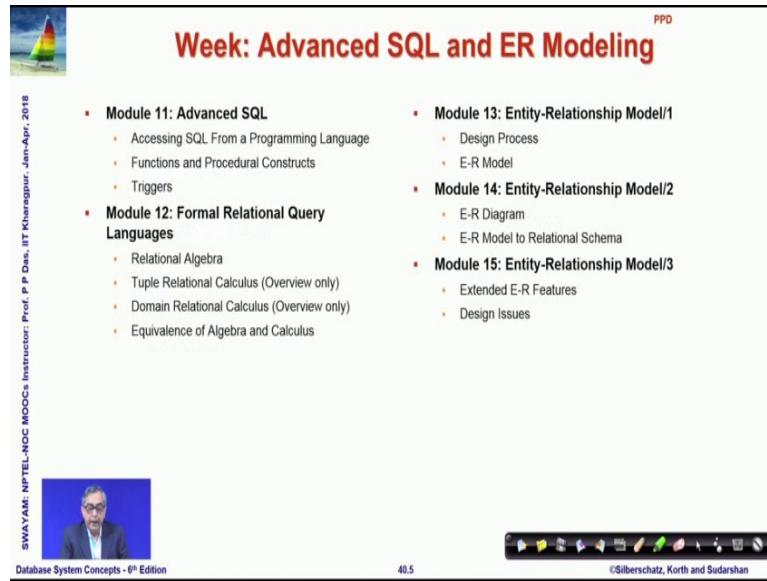
The slide title is "Week 02: SQL – Introductory & Intermediate". It features a small sailboat icon in the top left corner and the letters "PPD" in the top right corner. The slide content is organized into two columns of bullet points:

Module	Topics
Module 06: Introduction to SQL/1	History of SQL Data Definition Language (DDL) Basic Query Structure (DML)
Module 07: Introduction to SQL/2	Additional Basic Operations Set Operations Null Values Aggregate Functions
Module 08: Introduction to SQL/3	Nested Subqueries Modification of the Database
Module 09: Intermediate SQL/1	Join Expressions Views Transactions
Module 10: Intermediate SQL/2	Integrity Constraints SQL Data Types and Schemas Authorization

On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 8th Edition" and "40.4". On the right, there is a video thumbnail of a person speaking, a decorative toolbar, and the copyright notice "©Silberschatz, Korth and Sudarshan".

In week 2, we started off with query language SQL at an Introductory level and then at an Intermediate level which are really really the first major aspect of a database particularly relational database that a student must master.

(Refer Slide Time: 01:01)



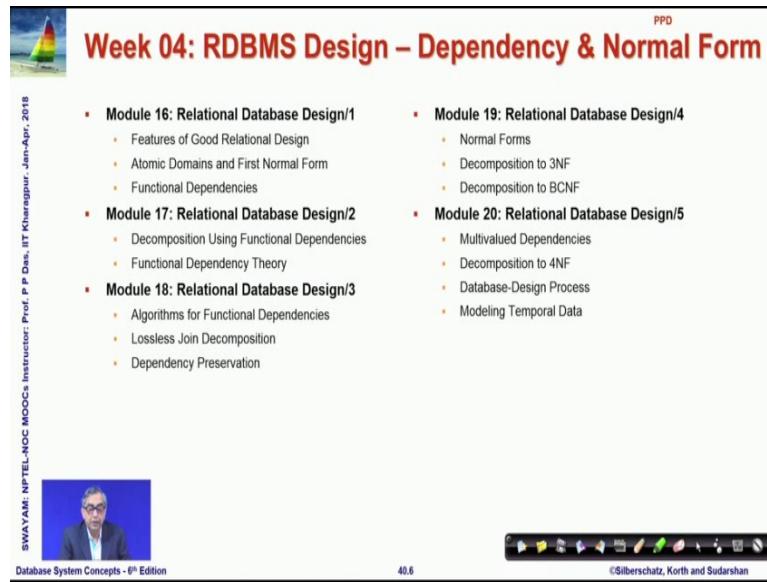
The slide title is "Week: Advanced SQL and ER Modeling". It features a small sailboat icon in the top left corner and a decorative bar at the bottom. The content is organized into two columns of bullet points:

<ul style="list-style-type: none">▪ Module 11: Advanced SQL<ul style="list-style-type: none">◦ Accessing SQL From a Programming Language◦ Functions and Procedural Constructs◦ Triggers▪ Module 12: Formal Relational Query Languages<ul style="list-style-type: none">◦ Relational Algebra◦ Tuple Relational Calculus (Overview only)◦ Domain Relational Calculus (Overview only)◦ Equivalence of Algebra and Calculus	<ul style="list-style-type: none">▪ Module 13: Entity-Relationship Model/1<ul style="list-style-type: none">◦ Design Process◦ E-R Model▪ Module 14: Entity-Relationship Model/2<ul style="list-style-type: none">◦ E-R Diagram◦ E-R Model to Relational Schema▪ Module 15: Entity-Relationship Model/3<ul style="list-style-type: none">◦ Extended E-R Features◦ Design Issues
---	--

On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a video thumbnail of the instructor, and at the bottom right are navigation icons and the text "Database System Concepts - 8th Edition" and "40.5". The bottom right also includes the copyright notice "©Silberschatz, Korth and Sudarshan".

In week 3, we continued with the advanced SQL and did the aspects of modeling from specification in terms of Entity Relationship Model.

(Refer Slide Time: 01:12)



The slide title is "Week 04: RDBMS Design – Dependency & Normal Form". It features a small sailboat icon in the top left corner and a decorative bar at the bottom. The content is organized into two columns of bullet points:

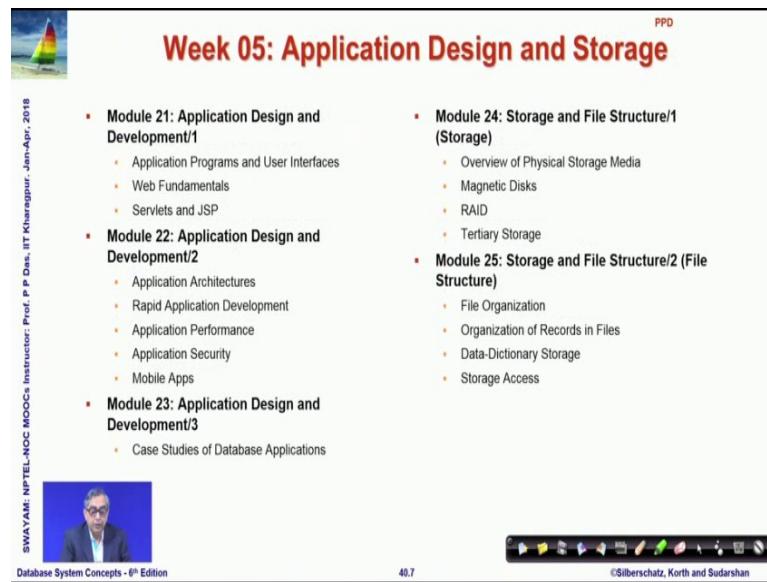
<ul style="list-style-type: none">▪ Module 16: Relational Database Design/1<ul style="list-style-type: none">◦ Features of Good Relational Design◦ Atomic Domains and First Normal Form◦ Functional Dependencies▪ Module 17: Relational Database Design/2<ul style="list-style-type: none">◦ Decomposition Using Functional Dependencies◦ Functional Dependency Theory▪ Module 18: Relational Database Design/3<ul style="list-style-type: none">◦ Algorithms for Functional Dependencies◦ Lossless Join Decomposition◦ Dependency Preservation	<ul style="list-style-type: none">▪ Module 19: Relational Database Design/4<ul style="list-style-type: none">◦ Normal Forms◦ Decomposition to 3NF◦ Decomposition to BCNF▪ Module 20: Relational Database Design/5<ul style="list-style-type: none">◦ Multivalued Dependencies◦ Decomposition to 4NF◦ Database-Design Process◦ Modeling Temporal Data
---	--

On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a video thumbnail of the instructor, and at the bottom right are navigation icons and the text "Database System Concepts - 8th Edition" and "40.6". The bottom right also includes the copyright notice "©Silberschatz, Korth and Sudarshan".

In week the next week, we did the design issues which was really the involved part and possibly the most important aspect of the relational database design beyond query coding query being able to write queries.

So, this is based on dependency and different normal forms and I am sure you have spent a good time on mastering these.

(Refer Slide Time: 01:40)



PPD

Week 05: Application Design and Storage

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

- Module 21: Application Design and Development/1
 - Application Programs and User Interfaces
 - Web Fundamentals
 - Servlets and JSP
- Module 22: Application Design and Development/2
 - Application Architectures
 - Rapid Application Development
 - Application Performance
 - Application Security
 - Mobile Apps
- Module 23: Application Design and Development/3
 - Case Studies of Database Applications
- Module 24: Storage and File Structure/1 (Storage)
 - Overview of Physical Storage Media
 - Magnetic Disks
 - RAID
 - Tertiary Storage
- Module 25: Storage and File Structure/2 (File Structure)
 - File Organization
 - Organization of Records in Files
 - Data-Dictionary Storage
 - Storage Access

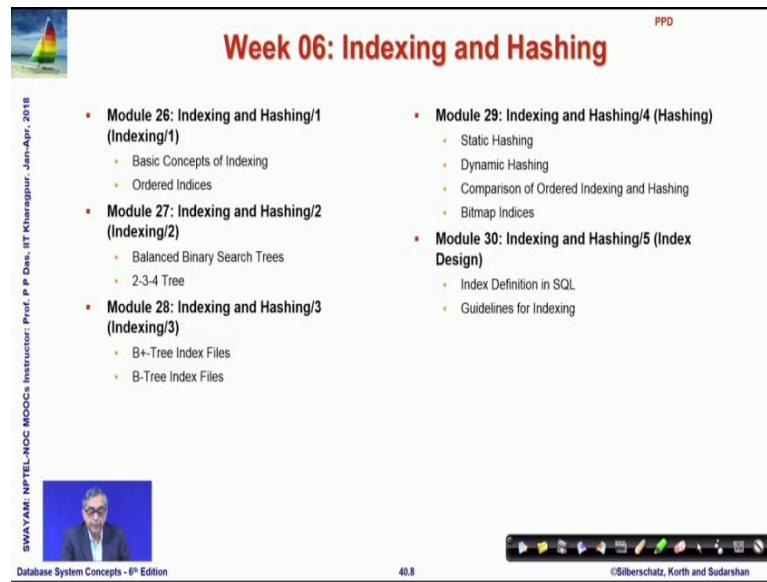
Database System Concepts - 6th Edition

40.7

©Silberschatz, Korth and Sudarshan

We followed up in week 5 with application design and discussing aspects of storage structure, how will actually the items, data items be stored in the different memory and disk structure.

(Refer Slide Time: 01:56)



PPD

Week 06: Indexing and Hashing

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

- Module 26: Indexing and Hashing/1 (Indexing/1)
 - Basic Concepts of Indexing
 - Ordered Indices
- Module 27: Indexing and Hashing/2 (Indexing/2)
 - Balanced Binary Search Trees
 - 2-3-4 Tree
- Module 28: Indexing and Hashing/3 (Indexing/3)
 - B+-Tree Index Files
 - B-Tree Index Files
- Module 29: Indexing and Hashing/4 (Hashing)
 - Static Hashing
 - Dynamic Hashing
 - Comparison of Ordered Indexing and Hashing
 - Bitmap Indices
- Module 30: Indexing and Hashing/5 (Index Design)
 - Index Definition in SQL
 - Guidelines for Indexing

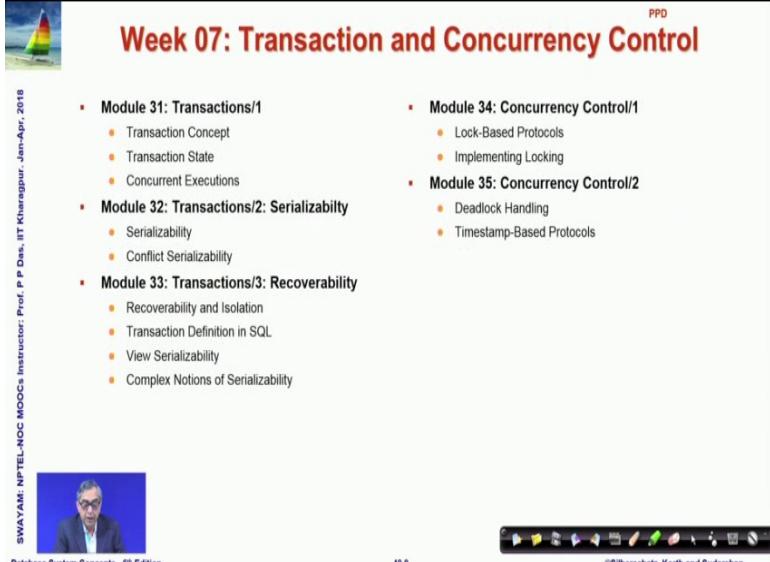
Database System Concepts - 6th Edition

40.8

©Silberschatz, Korth and Sudarshan

In the following week, in week 6, we discussed about indexing and hashing to for making the accesses really efficient.

(Refer Slide Time: 02:05)



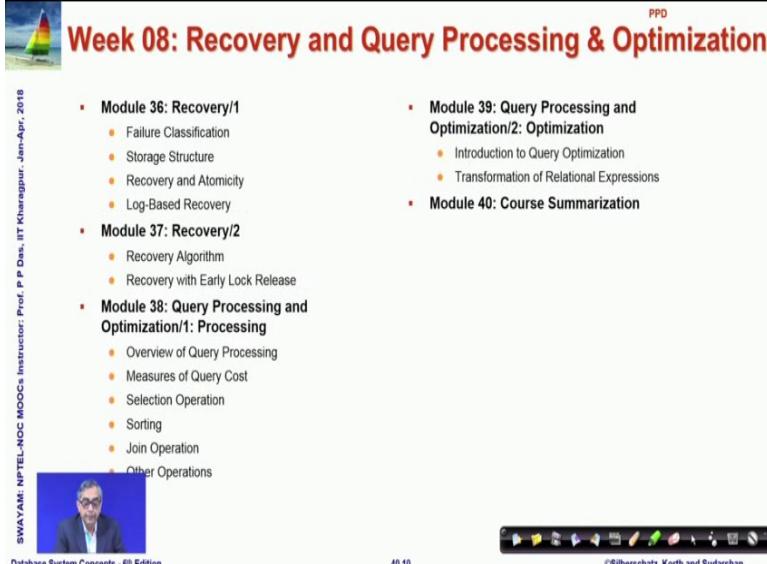
The slide is titled "Week 07: Transaction and Concurrency Control" in red at the top right. It features a small sailboat icon in the top left corner. The content is organized into two columns of bullet points:

<ul style="list-style-type: none">▪ Module 31: Transactions/1<ul style="list-style-type: none">○ Transaction Concept○ Transaction State○ Concurrent Executions▪ Module 32: Transactions/2: Serializability<ul style="list-style-type: none">○ Serializability○ Conflict Serializability▪ Module 33: Transactions/3: Recoverability<ul style="list-style-type: none">○ Recoverability and Isolation○ Transaction Definition in SQL○ View Serializability○ Complex Notions of Serializability	<ul style="list-style-type: none">▪ Module 34: Concurrency Control/1<ul style="list-style-type: none">○ Lock-Based Protocols○ Implementing Locking▪ Module 35: Concurrency Control/2<ul style="list-style-type: none">○ Deadlock Handling○ Timestamp-Based Protocols
--	---

At the bottom left, there is a small video thumbnail showing a man speaking. The footer contains the text "SWAYAM, NPTEL-NOC, MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018", "Database System Concepts - 8th Edition", "40.9", and "©Silberschatz, Korth and Sudarshan".

In week 7, we did another critical aspect of database systems that is how to make transactions work concurrently. So, we defined transactions and define what is Concurrency and then we took in two different critical aspects of Serializability that it is possible that we can execute transactions in a manner so that their instructions are intermixed, but then even in that case, they actually produce a result which is as if these transactions could have been executed in the serial order and we talked about the issues of recoverability in this respect and we specifically looked at different protocols, particularly two phase locking protocol for managing this kind of concurrency and the evils of deadlock that may happen when you do it concurrency and how simple protocols like time based protocol can handle that.

(Refer Slide Time: 03:03)



The slide is titled "Week 08: Recovery and Query Processing & Optimization". It features a sailboat icon in the top left corner and the PPD logo in the top right. The main content is organized into two columns of bullet points:

- Module 36: Recovery/1
 - Failure Classification
 - Storage Structure
 - Recovery and Atomicity
 - Log-Based Recovery
- Module 37: Recovery/2
 - Recovery Algorithm
 - Recovery with Early Lock Release
- Module 38: Query Processing and Optimization/1: Processing
 - Overview of Query Processing
 - Measures of Query Cost
 - Selection Operation
 - Sorting
 - Join Operation
 - Other Operations
- Module 39: Query Processing and Optimization/2: Optimization
 - Introduction to Query Optimization
 - Transformation of Relational Expressions
- Module 40: Course Summarization

At the bottom left is a small video thumbnail showing a man speaking. The footer contains the text "SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018", "Database System Concepts - 8th Edition", "40.10", and "©Silberschatz, Korth and Sudarshan".

And in the current week, we have dwelt with different strategies of recovery, particularly log based recovery and we have touched upon query processing and optimization.

So, this is you have got a very first level overview of this course. This is in a limited time and with limited number of assignments. So, you will just get a first level idea, this is not to make you really an expert of database systems, but this will certainly get you started well in terms of the database management programs, in terms of you are taking up advanced courses later on or in terms of actually taking up a job in different database area.

(Refer Slide Time: 03:48)

The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "SWAYAM, NPTEL-NOC MOOCs", "Instructor: Prof. P.P. Desai, IIT Kanpur", and "Jan-Apr., 2018". The main content area contains a bulleted list of objectives:

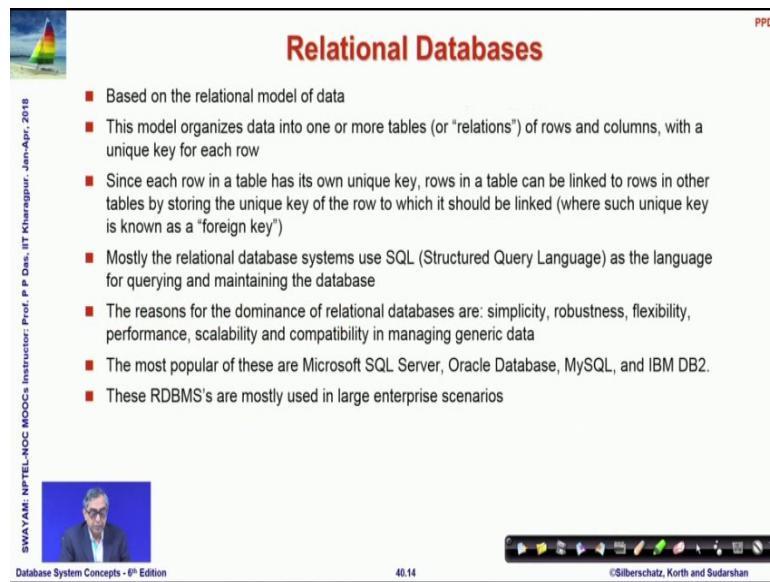
- The space of RDBMSs is crowded. We take a look into common RDBMS systems
- Non-Relational database systems are starting to dominate emerging applications. We present a brief overview
- What is the road forward? We outline likely job profiles in terms of a skills-profiles matrix and the companies to work for

At the bottom left is a small video thumbnail showing a person speaking. The bottom right shows the slide number "40.11" and copyright information "©Silberschatz, Korth and Sudarshan". A decorative toolbar is visible at the very bottom.

So, given that in this current module, I would discuss about few things beyond the textbook actually, the space of databases RDBMS bases are quite crowded, the lot of RDBMS bases you will see. So, I will just take a quick look in terms of the common RDBMS systems and there had been a number of queries on the forum and in the live session about that. I will also touch upon what we would like to discuss about non relational database systems which was not a part of the curriculum that we did here, but will just present a brief overview.

And then finally, I would like to conclude with what should be the road forward from you, presenting you a kind of a skill profile matrix so that what skills you must pick up to actually get a job off certain profile and what are the companies that you might look at working for.

(Refer Slide Time: 04:55)



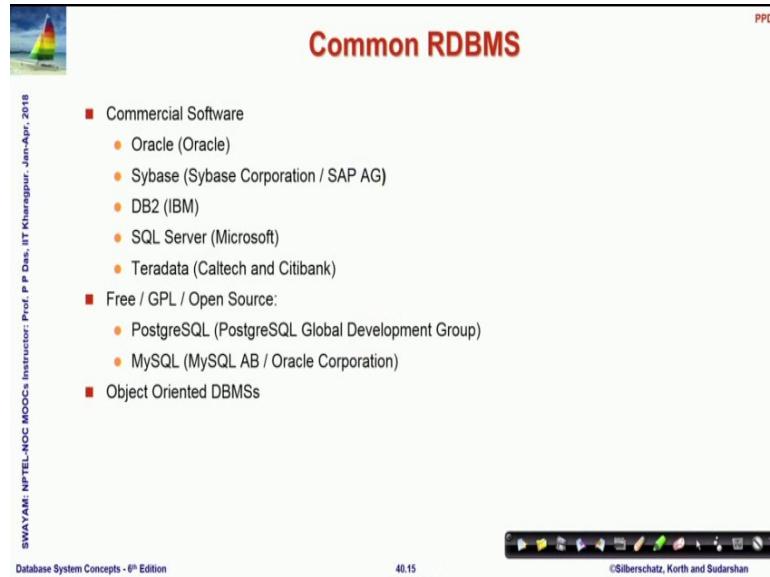
The slide is titled "Relational Databases" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main content area lists several bullet points about relational databases:

- Based on the relational model of data
- This model organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row
- Since each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a "foreign key")
- Mostly the relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database
- The reasons for the dominance of relational databases are: simplicity, robustness, flexibility, performance, scalability and compatibility in managing generic data
- The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.
- These RDBMS's are mostly used in large enterprise scenarios

At the bottom left is a small portrait of a man, and the bottom right shows a navigation bar with icons and the text "Database System Concepts - 8th Edition", "40.14", and "©Silberschatz, Korth and Sudarshan".

So, starting with the common databases, there are several relational database systems. So, in this slide you summarize basically what are the different aspects of relational database systems which you have been discussing so far. So, this is just a summary of that.

(Refer Slide Time: 05:07)



The slide is titled "Common RDBMS" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main content area lists categories of database systems:

- Commercial Software
 - Oracle (Oracle)
 - Sybase (Sybase Corporation / SAP AG)
 - DB2 (IBM)
 - SQL Server (Microsoft)
 - Teradata (Caltech and Citibank)
- Free / GPL / Open Source:
 - PostgreSQL (PostgreSQL Global Development Group)
 - MySQL (MySQL AB / Oracle Corporation)
- Object Oriented DBMSs

At the bottom left is a small portrait of a man, and the bottom right shows a navigation bar with icons and the text "Database System Concepts - 8th Edition", "40.15", and "©Silberschatz, Korth and Sudarshan".

Now, these are the common database systems. So, I have chosen the ones which are most widely used, most easily accessible and kind of large companies use them, large databases exist on them. So, there are primarily 2 classifications; one is a set of database

systems are commercial Oracle from the Oracle corporation, Sybase from Sybase corporation which is now SAP AG, DB2 from IBM, SQL Server from Microsoft and the recent entrant to that who is making a regular ripples is Teradata which is a you know joint database systems from Caltech and certain group of Citibank. So, if you are working for a company who subscribes to any of these database software, then you should be able to use them and understand what all you can do, but if you are working with smaller companies or you are working as a student, then you will need to use some of the database systems which are free or are on the GPL licensing or open source.

So, most prominent amongst them is PostgreSQL which is from a Postgres global development group. So, these are non commercial the software in the sense that you do not need to pay for them and they are on the GPL and some of some part of that would be open source as well and a very commonly used is MySQL which was originally from a Swedish company called MySQL AB, but now it is acquired by Oracle corporation, but it still does not you do not need to pay for that. So, these are the databases and systems to primarily look for and besides that there are some other database systems which use certain object oriented features on top of the relational features.

So, if you look through these, then in most cases you will find in terms of the gross functionality of the kind of SQL that you can write, a large subset of the SQL that you can write on databases maintained through these database systems will be same. So, what you have learnt here would be applicable irrespective of which database which of these database systems you are using, but of course there are specifics which would be different amongst them. So, in the next couple of slides, I have for on each one slide, I have given a brief background about the particular database system.

(Refer Slide Time: 07:48)

The slide has a header 'Oracle' and a small sailboat icon. The content lists features of Oracle, including its history, latest version, usage, languages, tools, and access methods. The footer includes the source 'Database System Concepts - 8th Edition', page number 40.16, and copyright information.

■ Multi-model commercial database management system produced and marketed by **Oracle Corporation**.
■ Larry Ellison, Bob Miner and Ed Oates started a consultancy called Software Development Laboratories (SDL) in 1977, and developed the original version of Oracle.
■ Latest Version: **Oracle Database 12c Release 2: 12.2.0.1 (patchset as of March 2017)**
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
■ Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)
■ Tools/ Editions: Oracle SQL Developer, Oracle Forms, Oracle JDeveloper, Oracle Reports for development of applications, Oracle Live SQL for test environment
■ Oracle can be accessed from Java through JDBC, Microsoft.NET through ODP.NET, C, C++ through OCI, ODBC, ODPI-C, Python through cx_Oracle

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

PPD

Database System Concepts - 8th Edition 40.16 ©Silberschatz, Korth and Sudarshan

So that you know you know how stable, how old or you know what are the basic nuances of that database system for example, Oracle started in 77. So, you can say, it is a it is a 40 year old database system. The latest version is 12 C and these are the different supports that it has.

(Refer Slide Time: 08:13)

The slide has a header 'Sybase' and a small sailboat icon. The content lists features of Sybase, including its history, latest version, languages, tools, and access methods. The footer includes the source 'Database System Concepts - 8th Edition', page number 40.17, and copyright information.

■ Relational model database server product for businesses developed by **Sybase Corporation** which became part of SAP AG.
■ Originally for unix platforms in 1987, Sybase Corporation's primary DBMS product was initially marketed under the name Sybase SQL Server.
■ Latest Version: **Sybase 16, released on 2014**
■ Languages: Sybase IQ, Transact-SQL
■ Tools/ Editions: Sybase SQL server for development of applications. Has a developer and express edition.
■ Sybase can be accessed from C, C++ through SQLAPI++, Java through JDBC

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

PPD

Database System Concepts - 8th Edition 40.17 ©Silberschatz, Korth and Sudarshan

Sybase also started in 1987. So, that is almost 30 years, but it is less you know less vibrant right now, the last stable released happen in 2014 about nearly more than 3 and a

half or 4 years ago and, but Sybase is a has been a very good database systems for programming through API's and it has really good support for that.

(Refer Slide Time: 08:42)

DB2

■ Db2 contains database-server products developed by **IBM**. Mostly relational models, but now includes object relational models
■ In 1970, Edgar F.Codd, researcher in IBM published the model for data manipulation.
■ Latest Version: **DB2 LUW 11.1 released on 2016**
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
■ Languages: Structured Query language (SQL), XML Query
■ Tools/ Editions: Advanced Enterprise Server Edition, Enterprise Server Edition, Advanced Workgroup Server Edition, Workgroup Server Edition, Direct and Developer Editions and Express-C.
■ Db2 can be accessed from C, C++, Java, Ruby, Perl through a package of DB2 API's

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

Database System Concepts - 8th Edition

40.18

©Silberschatz, Korth and Sudarshan

DB 2 is also a very old possibly the oldest surviving database systems which started in 1970. So, when almost the E. F Codd of the Boyce Codd normal form published the data manipulation schemes from IBM. So, this is also a widely used, last release 2016.

(Refer Slide Time: 09:07)

SQL Server

■ Relational database management system developed by **Microsoft**.
■ SQL Server 1.0, a 16-bit server for the OS/2 operating system in 1989
■ Latest Version: **SQL Server 2017**
■ Used for running online transaction processing (OLTP) and online analytical processing (OLAP)
■ Languages: Transact SQL
■ Tools/ Editions: Enterprise, Standard, Web, Business Intelligence, WorkGroup, Express
■ SQL server can be accessed from Java through JDBC, C, C++ through ODBC

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

Database System Concepts - 8th Edition

40.19

©Silberschatz, Korth and Sudarshan

Microsoft started database systems in 1989, last release happened last year and that is very widely used if you are particularly on windows system, it is one of the most popular one in terms of the windows operating system.

(Refer Slide Time: 09:24)

■ Relational database management system developed by Caltech and Citibank's advanced technology group
■ In 1984, the first version of Teradata was released
■ Latest Version: **Teradata 15**
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
■ Languages: BTEQ(Basic Teradata query)
■ Tools/ Editions: Developer Edition, Express Edition
■ SQL server can be accessed from Java through JDBC, C, C++ through ODBC

Teradata is relatively new. It was released in 1984 and it is, but it is one where lot of new developments are still happening and new experiments keep on happening and the current version is a Teradata 15.

(Refer Slide Time: 09:43)

■ Open source relational database management system produced by PostgreSQL Global Development Group, a diverse group of many companies and individual contributors.
■ First version in 1986 by researchers of POSTGRES project
■ Latest Version: **PostgreSQL 10.3 released on 2018**
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads, Supports big data analytics
■ Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)
■ Oracle can be accessed from Java through JDBC, Microsoft.NET through npgsql, C, C++ through libpq,

And in terms of the ma free or open source GPL databases Postgres started in 1988 about 30 years back and as I said, this is this is has a release even half of this year.

(Refer Slide Time: 10:04)

MySQL

- Open source relational database management system produced by Swedish company MySQL AB, now owned by Oracle Corporation
- First internal release on 23 May 1995
- Latest Version: MySQL 8.0.4 released on 2018
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: Structured Query language (SQL), Procedural SQL (PL-SQL)
- Oracle can be accessed from Java through JDBC, Microsoft.NET through ADO.NET, C, C++ through ODBC

SWAYAM-NIETL-NOC-MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8th Edition

40.22

©Silberschatz, Korth and Sudarshan

So, it is a very vibrant system. MySQL probably most widely used amongst the free community among the open source community also where the first internal release happened in 1995. The recent releases happened this year. So, these are the common database systems that you will come across. So, I mean given the organization that you are working with, first find out which database system it uses and then look into the specific manual for that and specific features.

(Refer Slide Time: 10:34)

Object-oriented DBMS (OODBMSs)

- Combines database capabilities with object oriented programming language capabilities
- OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS
- Objects have a many to many relationship and are accessed by the use of pointers.
- Access to data can be faster because an object can be retrieved directly without a search, by following pointers.
- Most object databases also offer some kind of query language, allowing objects to be found using a declarative programming approach.
- Examples:
 - Objectivity/DB,
 - O2,
 - Object Store

Ref: https://en.wikipedia.org/wiki/Object_database

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

Database System Concepts - 8th Edition

40.23 ©Silberschatz, Korth and Sudarshan

Beyond these a Relational Database Systems also, there are certain database systems which use Object oriented notions in that. So, if you are familiar with object orientation then you would have understood that the relational approach does not make keep things object oriented because you are always flattening out in terms of attributes and you are trying to look at the attributes, but it went for example, when you want to model the same thing in terms of a C++ or java program, you would like to look at a course as an as an object, as a class you would like to look at instructor as a class, you would like to look at teaches as a as a kind of class and their instances. So, there has been attempts to make give a object orientation kind of layer on top of relational databases or define things in that way. Objectivity DBO 2 objects store are some of the examples, but unfortunately this is have not been as popular as a regular relational databases.

So, if you happen to use any one of them, then you should be cause a cautious that you know you really know why you are using it and you would be able to go a long way in terms of that.

(Refer Slide Time: 11:53)

The slide is titled "Parameters" in red at the top right. On the left, there is a small image of a sailboat on water. The background is white. At the bottom, there is a decorative footer bar with various icons.

Parameters

- We compare the RDBMSs based on the following parameters:
 - OS support
 - Fundamental features
 - Limits
 - Tables and views
 - Indexes
 - Database capabilities
 - Data types
 - Other objects
 - Partitioning
 - Access control
 - Programming Language Support

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8th Edition

40.24

©Silberschatz, Korth and Sudarshan

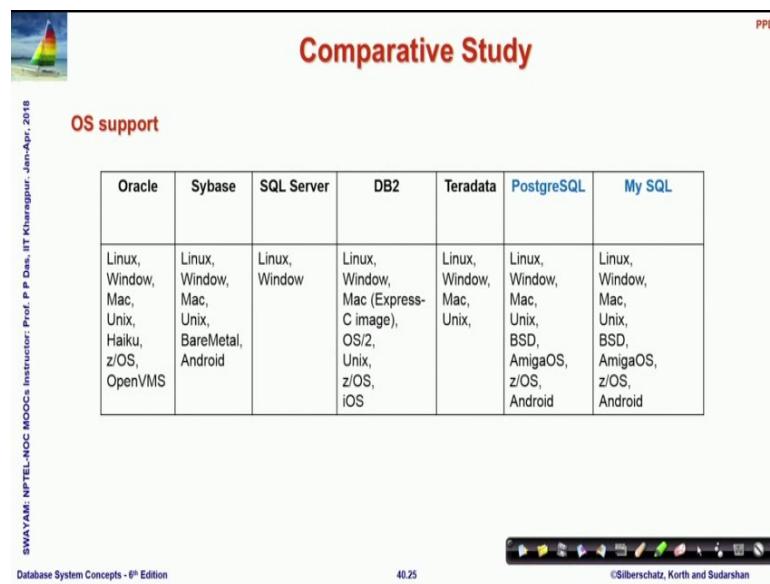
Now, when you come across a particular system that your company or your university needs to use and you would like to choose, then it will be good to look at the different aspects of that system. These are here are some of the parameters on which these database systems vary in a minimal to a very large extent, in terms of what operating systems it supports, what are the fundamental features, what are the limits for example, every database sets a number of limits in terms of the index size, in terms of the table size and whole lot of that.

How are the tables and views created what the kind of restrictions you have that, what kind of indexes has support the capabilities the data types, the different databases support. We have talked about a very limited set of data types in terms of SQL, but in an actual commercial or even you know open source database, the data types could be wider than that what kind of other objects partitioning access control mechanism. Access control is very important for ensuring security and finally, what kind of programming language support do you have.

Because as we have seen in the application development module, that it is not enough to just have a you know the database firing SQL queries, no application user will actually fire SQL queries. The application user needs and in GUI possibly or a text interface through which it will put queries in a different form and that needs to be processed by taking it to the database engine. So, you need possibly an interface which is in terms of

C, C++, Java, Python, this kind of programming language. So, how do you connect to or embed such embed your relational query into different languages that differ between different database systems. So, these are the parameters that you must look at. In the next series of slides, which I will not you know discuss really because the these are more like data.

(Refer Slide Time: 14:03)



Comparative Study

OS support

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Linux, Window, Mac, Unix, Haiku, z/OS, OpenVMS	Linux, Window, Mac, Unix, BareMetal, Android	Linux, Window	Linux, Window, Mac (Express-C image), OS/2, Unix, z/OS, iOS	Linux, Window, Mac, Unix,	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android

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

Database System Concepts - 8th Edition

40.25

©Silberschatz, Korth and Sudarshan

But here after given a compilation of different you know on different aspects, how do these common database RDMS systems agree or differ. So, this is like a slide which shows what are the operating system support for different databases. So, if you are for example, working on android, then you can easily make out that you do not have a choice to use SQL server or to use Oracle, but you can use Sybase.

But you can use Postgres and MySQL actually, if you look into these two columns, right most columns which are for the open source databases, you will find that they have the widest choice in terms of operating system. In many aspects you will find that these free database systems have a better you know options for you; obviously, when it comes to you know really the core, core of database systems in terms of really really supporting very large databases, really really supporting very fast operations, really really supporting very secure applications, you might need to only work with commercial software because they offer that, but otherwise for a large number of common you know

medium scale or low cost applications the free database systems, Postgres and MySQL are really good options there are different such features.

(Refer Slide Time: 15:23)



Comparative Study

PPD

Basic Features

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports ACID properties for transactions, implicit commit for DDL, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, hash and partition for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control

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

Database System Concepts - 8th Edition

40.26

©Silberschatz, Korth and Sudarshan

The basic features are compared to here.

(Refer Slide Time: 15:27)



Comparative Study

PPD

Limits

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Max DB Size: 2PB	Max DB Size: 104TB	Max DB Size: 524,272 TB	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited
Max Table Size: 4GB * block size	Max Table Size: File size	Max Table Size: 524,272 TB	Max Table Size: 2 ZB	Max Table Size: Unlimited	Max Table Size: 32 TB	Max Table Size: 256 TB
Max Row Size: 8KB	Max Row Size: File size	Max Row Size: 2TB	Max Row Size: 32,677 B	Max Row Size: 64 GB	Max Row Size: 1.6TB	Max Row Size: 64KB
Max Column per Row: 1,000	Max Column per Row: 45,000	Max Column per Row: 1,024	Max Column per Row: 1,012	Max Column per Row: 2048	Max Column per Row: 1600	Max Column per Row: 4096
Max CHAR size: 32,767 B	Max CHAR size: 2GB	Max CHAR size: 2GB	Max CHAR size: 32 KB	Max CHAR size: 64,000 bits	Max CHAR size: 1GB	Max CHAR size: 64 KB
Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 38 bits	Max Number size: Unlimited	Max Number size: 64 bits
Max Column Name size: 128		Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 63	Max Column Name size: 64

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

Database System Concepts - 8th Edition

40.27

©Silberschatz, Korth and Sudarshan

At compile different limits of different database systems here. So, you can see in terms of maximum row size, columns per row and so on and so forth, how do they differ. So, if you are making a choice in terms of what database I will use.

(Refer Slide Time: 15:42)

The slide title is "Comparative Study" with a subtitle "Tables and Views". It contains two tables comparing Oracle, Sybase, SQL Server, DB2, Teradata, PostgreSQL, and MySQL.

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables (apart from basic)

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Static+Dynamic	Static	Static	Static+Dynamic	Static	Static	Static

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

Database System Concepts - 8th Edition 40.28 ©Silberschatz, Korth and Sudarshan

You can use these information. This talks about tables we use the type systems, what kind of typing is used.

(Refer Slide Time: 15:48)

The slide title is "Comparative Study" with a subtitle "Data Types". It contains two tables comparing Oracle, Sybase, SQL Server, DB2, Teradata, PostgreSQL, and MySQL.

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; And other miscellaneous types like Spacial, Image, Audio, Dicom, Video	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; And other miscellaneous types like Money	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Date/Time; Period, Interval, Geometry, xml, json	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Boolean	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit

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

Database System Concepts - 8th Edition 40.29 ©Silberschatz, Korth and Sudarshan

The different data types that are used are given here.

(Refer Slide Time: 15:52)



Comparative Study

PPD

Indexes

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports only Basic B/B++ indexes	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports Hash, Partial, Bitmap, Reverse	Supports R/R++, Hash, Partial, Bitmap, Reverse	Supports R/R++, Hash,
Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes	Apart from Basic B/B++ indexes

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

Database System Concepts - 8th Edition 40.30 ©Silberschatz, Korth and Sudarshan



The index mechanisms are discussed here.

(Refer Slide Time: 15:57)



Comparative Study

PPD

Database Capabilities

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clobs, Common Table Expressions, Windowing Functions, Parallel Query	Supports Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Blobs and Clobs, Common Table Expressions	Supports Union, Outer Joins, Except, Inner Selects, Blobs and Clobs, Common Table Expressions

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

Database System Concepts - 8th Edition 40.31 ©Silberschatz, Korth and Sudarshan



The capabilities of the database, what it can do overall.

(Refer Slide Time: 16:01)



Comparative Study

PPD

Other Objects						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Cursor, Trigger, Function, Procedure, External Routine

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

Database System Concepts - 8th Edition 40.32 ©Silberschatz, Korth and Sudarshan



Other kinds of objects that it supports.

(Refer Slide Time: 16:05)



Comparative Study

PPD

Partitioning						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Range, Hash, Composite, List	none	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List

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

Database System Concepts - 8th Edition 40.33 ©Silberschatz, Korth and Sudarshan



The different partitioning mechanism.

(Refer Slide Time: 16:08)



Comparative Study

PPD

Access Control						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Enterprise Directory compatibility, Patch Access
						

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

Database System Concepts - 8th Edition

40.34

©Silberschatz, Korth and Sudarshan

The access control which is critical for ensuring good security and good management of that database are given here.

So, these are kind of the different across different features, this is parameters, this is how they compare. So, well this is for your you know reference only, this is this is not for your assignment or examination, but I just wanted to have a view that with all the theory and you know a small hands on that you have seen, when you go to the real life what are the expected things what are the expected system this will have to work with. Now let us just move on and let us let me just briefly talk about the a group of database systems which are non relational and of course I would warn you that the basis for these DBMS is are not covered in this course, is beyond the course, but just for your information and to keep in keep you in tune with what is happening frequently around the industry today.

(Refer Slide Time: 17:16)

What is Big Data?

- Big data is data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them
- Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source
- **5V's (characteristics) of big data:**
 - **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.
 - **Variety:** The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
 - **Velocity:** In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.
 - **Variability:** Inconsistency of the data set can hamper processes to handle and manage it.
 - **Veracity:** The data quality of captured data can vary greatly, affecting the accurate analysis

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

So, the non-relational database systems have arisen from what they all have must have heard of is the whole aspect of Big Data. Big Data as a name suggests is certainly voluminous data, complex data and now the question that you might have is if I have done a good relational design, if I have a good RDMS, can I not handle big data?. The question here really is, big data is not only about volume, the volume is only one aspect which is really large. So, big data typically are characterized by certain Vs.

So, these are not any very standardized characterization, but these are more commonly accepted once. So, one is volume, that the quantity of data when a for a big data situation has to be very very large. Now again what is a very very large is again a subjective question, some you can say that a million record is large, someone else would say no million record is small, it is actually 10 million is large; some might say that it is a database need to be petabytes to be large and so on.

But these are all subjectivity, but it is large has a voluminous existent in certain sense, there has to be different variety different types of data. So, all that we have seen in the relational database is basically your you know strings and numbers if you look at it in different ways we are seeing, whatever we have dealt with in all these through all this 39 modules so far, they are primarily about strings and numbers, but nothing else we have not, but big data can be about free text, it could be about natural language comments your regularly writing comments on your Facebook.

So, those Facebook comments are phrases and if I want to make certain query based on that, if I want to make a query that, how many Facebook users has commented on the success of Virat Kohli as a captain of India if I want to make such a query, then the question is how do I do that? Because it is not something where you have a at the information in a very structured way this is no there is no relational schema which says that well the values are put in terms of Virat Kohli having done very good, moderately or marginally or you know the captain Indian captains are successful, not successful and so on.

These are this happen in terms of various texts, phrases, clauses that we write. So, variety is a major issue then, it could have word your images video. So, big data includes all of that. The third V is about velocity that the processing speed may need to be really really fast, often in big data often we say that the processing has to be real time which means what is real time. Real time is basically that from the time I fire the query and to the time I get the result, there is a fixed time limit within which it has to happen.

So, if I if I if I really want to do a railway reservation that also is a kind of real time, but that is not very critical because it is if I get the reservation done in one minute, it is also ok, if it takes 5 minutes, it is good if we can happen in 10 seconds, but I do not ever need it in say 20 millisecond. So, but in when you talk about real time, it could really be about getting all these processing done in millisecond, microsecond, nanosecond and so on. And those kind of real time systems with a large volume of varied data, it is a big challenge.

So, those are the challenges of big data, then there could be variability inconsistency of the data that you are because maintaining integrity is a big problem; there could be issues in terms of quality of the data that is called veracity. So, actually these things characteristics that I have put, there are lot of debates in terms of that or many people take these 3 and say that there these are the 3 V s of big data. There is a 3 main characteristics, but off let more and more people are also considering variability and veracity are as the characteristics of big data. Now as it happens, is if you look into these requirements and what you have you have a fairly good idea of relational databases now what they can do, how to design them, how to query them, how to implement them, you will understand that it is not easy to meet these requirements using the relational model.

(Refer Slide Time: 22:07)

Why do we need non-relational databases?

To effectively support Big Data

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

Database System Concepts - 8th Edition 40.37 ©Silberschatz, Korth and Sudarshan

So, we need their non-relational databases to effectively support big data. That is that is a one major reason that you need big data.

(Refer Slide Time: 22:16)

Non-Relational Databases

PPD

- Does not follow the relational model
- Offer flexible schema design
- Handle unstructured data that does not fit neatly into rows and columns
- Typically Open Source
- Scalable using cheap commodity servers
- The popular ones are:
 - MongoDB, DocumentDB, Cassandra, Couchbase, HBase, Redis, and Neo4j
- These databases are usually grouped into four categories:
 - Key-value stores
 - Graph stores
 - Column stores, and
 - Document stores

Non-Relational Databases are also known as NoSQL Databases

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

Database System Concepts - 8th Edition 40.38 ©Silberschatz, Korth and Sudarshan

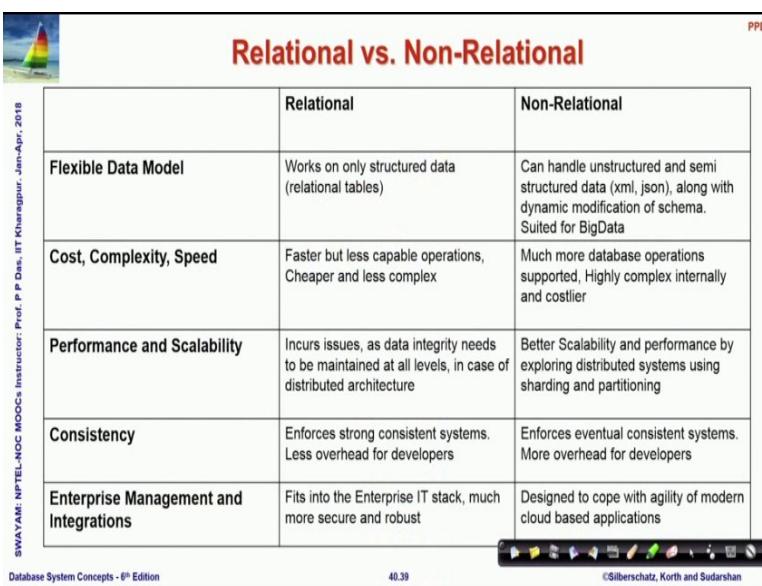
So, that in non-relational databases certainly as the name suggests, do not follow relational model, they offer flexible schema design. The schema may itself change while the database is evolving which is not the case in the relational schema is fixed, only the data can change, but here the schema itself can change. It may be able to handle unstructured data, make natural language comments like images like audio coming in

which do not fit nearly into your you know table structure, some of the other feature sites they are typically open source because you know still in an experimental stage and needs to be scalable and some of the popular ones are, these are the names you must have heard about at least some of them like MongoDB like Cassandra like HBase and so on.

Now again in terms of the non relational database, it does differ in terms of all non relational database error are not of the same type, there are there have been 4 different styles or strategies to actually generate realize these non relational databases, they are called key value store graph, store columns stores and document store. They are also known as no SQL databases. I, I personally find the name no SQL a little misnomer, it no SQL does not mean that you are strictly prohibited from not using SQL in these databases.

But I would rather like to read it more as no SQL means that it is not only SQL like in a relational database, you can use only SQL and solve problems; here you need to do lot of other things beyond that.

(Refer Slide Time: 24:00)



The slide features a title 'Relational vs. Non-Relational' in red font, a small sailboat icon in the top left, and a 'PPD' watermark in the top right. The main content is a comparison table with five rows. The first row is a header with two columns: 'Relational' and 'Non-Relational'. The remaining four rows compare specific characteristics:

	Relational	Non-Relational
Flexible Data Model	Works on only structured data (relational tables)	Can handle unstructured and semi structured data (xml, json), along with dynamic modification of schema. Suited for BigData
Cost, Complexity, Speed	Faster but less capable operations, Cheaper and less complex	Much more database operations supported, Highly complex internally and costlier
Performance and Scalability	Incurs issues, as data integrity needs to be maintained at all levels, in case of distributed architecture	Better Scalability and performance by exploring distributed systems using sharding and partitioning
Consistency	Enforces strong consistent systems. Less overhead for developers	Enforces eventual consistent systems. More overhead for developers
Enterprise Management and Integrations	Fits into the Enterprise IT stack, much more secure and robust	Designed to cope with agility of modern cloud based applications

Navigation icons are located at the bottom right of the slide.

So, here is a quick comparison between the relational and the non-relational, in terms of the flexibility of the data model, relational is very structured when on relational has to have handle unstructured data, semi structured data and therefore it has to be flexible in terms of the data model, cost complexity and speed faster less capable, but cheaper and less complex, but in non-relational, you are talking about much more database operations

highly complex in internal structure usually costlier, performance and scalability certainly non-relational ones need to be better scalable, consistency have a very strict consistency rules in relation, but in non-relational you use some kind of you know eventual consistent system. So, maybe not always not everything is consistent in that way, enterprise management and integration, relational fits very well into because it is been around for as you have seen the little bit of history of all these common databases, it is more than 40 years that they have been around.

So, they easily fit into the IT stack whereas, non-relational is still on the in the agile form of development that is becoming more and more common it fits into the cloud based development and so on. So, these are some of the distinctions that exist.

(Refer Slide Time: 25:27)

Types of NoSQL Databases

1. **Key-value stores:**
 - These type of databases work by matching keys with values, similar to a dictionary. There is no structure nor relation. **Example:** Redis, MemcacheDB
2. **Graph stores:**
 - These use tree-like structures (i.e. graphs) with nodes and edges connecting each other through relations. **Example:** OrientDB, Neo4J
3. **Column stores:**
 - Column-based NoSQL databases are two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and unstructured data to be kept and used (e.g. a record with tons of information). **Example:** Cassandra, Hbase
4. **Document stores:**
 - These DBMS work in a similar fashion to column-based ones; however, they allow much deeper nesting and complex structures to be achieved (e.g. a document, within a document, within a document).
Documents overcome the constraints of one or two level of key / value nesting of columnar databases. **Example:** MongoDB, Couchbase

And these are the different types of no SQL databases, there is a key value store strategy Redis and MemcacheDB follow this strategy, graph store is used by orient DB and Neo4J; column store is used by Cassandra and HBase document store, MongoDB, Couchbase, they use document store. So, these are I am in this is not just about going a deeper into what they are or how they are distinguished, I just want you to have an idea that well. These are different from the relational databases they can do lot of structured, handling of unstructured data they can actually use a scalability of volume a variety which is relationships cannot do and, but they have there are different principles for actually implementing them and there is a deeper.

So, if you are interested, you can take specific courses which deal with the big data and prepare yourself for the bigger challenges ahead.

(Refer Slide Time: 26:29)



Comparative Study

PPD

When to Use

Redis	MemcacheDB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Caching, Queuing frequent information, Keeping Live information, Supports lists, sets, queues and more	Caching, Queuing frequent information, Keeping Live information,	Handling complex relational information, Modelling and handling classifications	Handling complex relational information, Modelling and handling classifications	Keeping unstructured non-volatile information, useful for content management	Keeping unstructured non-volatile information, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management

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

Database System Concepts - 8th Edition 40.41 ©Silberschatz, Korth and Sudarshan

I have also done similar to the relational database, I have presented here a tentative comparative study between these different non no SQL databases in terms of what is the context in which you use them.

(Refer Slide Time: 26:46)



Comparative Study

PPD

Handling Relational Data

Redis	Memcache DB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Supports ACID, query only on key	Supports ACID, query only on key	Supports ACID and joins	Supports ACID and joins	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries, Nesting Data	Supports ACID, Multiple Queries, Nesting Data

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

Database System Concepts - 8th Edition 40.42 ©Silberschatz, Korth and Sudarshan

Or now while you are doing this unstructured data handling, there will be lot of data which is also structured.

So, how do you, along with this know SQL, how do you handle the relational data with these?

(Refer Slide Time: 26:59)



Comparative Study

PPD

Performance							
Redis	Memcache DB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Highly Scalable, Highly Flexible, not complex in terms of representation and use	Highly Scalable, Highly Flexible, not complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use
SWAYAM-NPTEL-NOCS Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018							

Database System Concepts - 8th Edition 40.43 ©Silberschatz, Korth and Sudarshan

Databases what how do how do the performance compare between these different no SQL databases and based on all that you can make some judgment and it is it is very important to in today's time naturally knowing relational databases the foundational ones are very important, but it is always good to look forward be with the time and I will urge that if you have started growing interest in handling of data do take specific courses on big data and no SQL databases. I will end this discussion with a very simple skill job profile matrix which Will give you some idea in terms of, it will you can use it for a certain kind of self-assessment as well.

(Refer Slide Time: 27:37)

Skills	Years of Experience	Under-standing Specs and Schema	Coding Query In SQL (DML)	Analyzing Specification, Schema Design and Normalization (DDL)	Application / Database Architecture, Indexing, Performance Optimization	Database Administration	Databases, Algorithms, Architecture, Compilers, ...	Data Mining, Machine Learning, Big Data, Programming
Job Profiles								
Application Programmer	0-4	X X						
Senior Application Programmer	2-6	X X X X	X					
Database Analyst / Architect	4-8	X X X X	X					
Database Administrator	8-10	X X X X X	X		X			
Database Engineer (multiple grades)	2-10	X X X X X	X		X X			
Big Data Programmer, Analyst	0-6	X X X X X					X	

So, let me just explain the structure of this matrix, what I have tried to do is here I am sorry. Here on the left, I have given different typical job profiles this is if you look into LinkedIn, Naukri and all that you will find these kind of profiles being. So, then at the lowest level there are application programmers for which typically 0 to 4 years of experience are asked for.

Then the next level, this is so, this is your kind of your career progression also. If you if you choose to take up databases as your primary job profession, this next level is a senior application programmer which requires 2 to 6 years of experience depending on the organization and depending on your skills. Then you move on to database analyst or architect which you happen in 4 to 8 years of time and on a little different track because these are these are primarily in terms of application development and hierarchy on that and the other is an administrator track who actually administers the database in an organization, controls all the all that is happening in different database applications, typically 8 to 10 year's experience is required.

And some of that, so this these are the about actually in terms of you know profiles that are related to applications and this is a profile which is related to, if you really want to become a database engineer in a sense that you want to you know make changes in Oracle, you want to make changes in say MySQL, you want to make changes in say MongoDB or say that relation will say the Sybase.

So, if you want to become a database engineer who changes the database system itself or develops the database system itself, then this is the kind of background you will need. There is a kind of number of years, you would need and of course it is not a single grid there are multiple grades, you know junior and mid levels in here and those kind and last which have shown in different color are the whole set of profiles which relate to programming the big data, analyzing the big data and so on.

We are at present, it is companies are typically asking for 0 to 6 years of experience depending on actual skills that you have. On this side, I have shown a whole grouping of skills, this is the first basic level that you must understand specs and schema, without that you cannot do any of this and you must have a skill for coding in inquire in SQL, the DML part significantly, without that as you can see you cannot pick up any of these profiles whereas, if you go a little if you go little senior you know gaining experience and you should be able to analyze specs that is, you should be able to design schema do normalization and get into this.

So, at a very initial level, you may not be expected to do all of that schema design and normalization by yourself, but it would be good to be able to do that, but well there will be seniors to help you, but if you once you become a senior application programmer that becomes onward that becomes a critical skill to have. Then the next level would be in terms of application or database architecture management deciding on how to index, performance optimization.

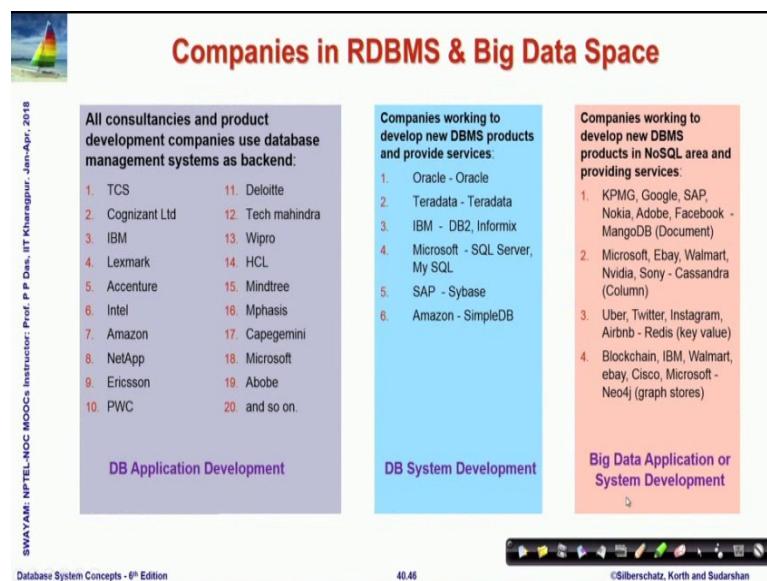
So, between these two, there will be certain overlaps a senior application programmer in addition to doing this might do some of these optimization techniques depending on how competent he or she is. Or some database architect may focus only on this, but these are the typical skills that you need. But to be a database administrator, you need all of these skills, but you are specifically administering a certain organizations enterprises whole database system. So, it is just not one database application, but a whole lot of databases and whole lot of user groups, security, network connectivity and all that.

So, that needs certainly bigger experience it can, you can see that experience level is much higher and the skill sets. If you want to become a database engineer, that is not focus only on the application side, but also have some more understanding in terms of actually doing working in the internals of the database systems, then you need whole lot

of additional skills like good knowledge and algorithms, in architecture, in compiler all of that; only then and coupled with coupled with all the database knowledge, then you will be able to work as a database system engineer. And in the emerging areas of what is big data where, you need to have now of course, the I am saying this is 0 to 6, it could be 0 to 8 kind of, not more than that because it did not exist quite a long time ago, but you need to have a basic level of at least this much of the relational database understanding and knowledge, but what is critical is a whole set of other skills like, you must be aware with big data the data mining, warehousing strategies machine learning or is often very useful in this kind of big data applications, python programming, tensor flow all these become critical. You have to be a good programmer in any case I mean not only just an SQL program and you might have to be a good program and in C or C plus plus or python of these, but that is it that is a very very emerging area.

So, if you can acquire a little bit of besides database you know he said that the basics of the database along with that if you pick up few basics or from here, you will be able to enter into the space and that will give you a very very bright future in my view otherwise you can focus on the application programming stat as I have mentioned. So, this is the basically skill profile matrix that you have mapping that you have that you can focus on.

(Refer Slide Time: 33:49)



So, finally, before I close here a glimpses of companies that are in the very active in the RDMS space really really any big organization you talk about, they have consultancy

projects, product development different database management back end services and so on. So, DB application development, I have listed some around 20 companies, but there are really 100s of them almost. Any big organization in any area you think of, they require databases. So, in terms of beta based application programmer and senior programmer and to some extent architect, you have a wide range of jobs available which you may just grab; if you have been able to study write the basics of the database. The second group of companies which I show here, these are system development companies who are actually working on the new DBMS products and services around that.

So, these are companies like Oracle, Teradata or Microsoft and so on, naturally these are big companies and you need more lot of more skills besides the database like I said algorithms programming and all that to crack a job here and here are some of some companies which I have mentioned, but there are many others who are focusing on the big data space.

So, I have tried to you know these may not be absolutely accurate because you know these are all collected from different sources, but these are the different companies and the kind of non relational database that they are focusing with working with. So, if you pick up certain skills in those in a certain non-relational no SQL database, then you can target the corresponding companies better or other companies and you can see that all. You know new generation companies, the companies were working for products for the next 10, 10, 15 years are in this space.

So, there are whole lot of opportunities for you all if you if you prepare a little hard, then you will I mean job will run after, you will not have to run after the job.

(Refer Slide Time: 35:56)

The screenshot shows a presentation slide with the title "Final Words" in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of five items:

- Read the DBMS Text book thoroughly and solve exercises
- Practice query coding
- Practice database design from specs
- Besides DBMS, develop good knowledge in programming, data structure, algorithms and discrete structures
- Seek help, if you need to – mail us

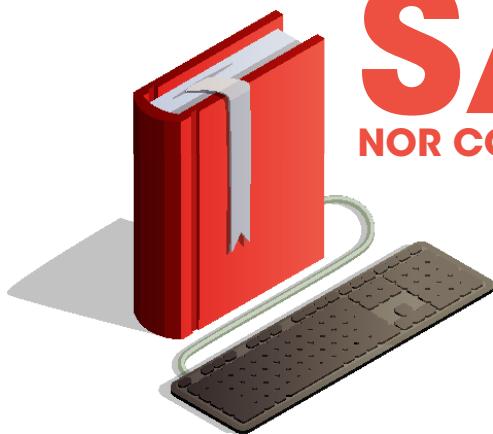
On the left margin, vertical text reads: "SWAYAM, NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a small video thumbnail of a man speaking. The bottom right corner shows "40.47" and "©Silberschatz, Korth and Sudarshan".

So, with that I conclude this course a couple of final words the hygiene words. Read the DBMS textbook thoroughly and solve exercises. There is no shortcut to that, there is no other way to master the horse other than this you must practice query coding as much as you can, practice database design from specification. We are releasing a tutorial on this where for a hospital management system we are showing from the specification how you can do the initial schema and the refinements and finally, how can you implement it using my SQL.

So, do similar practices very heavily. Keep in mind the database the knowledge of database system alone will not be good enough to get a good job, get a good placement. So, develop good knowledge in programming data structure, algorithms and discrete structures; these are the minimum required around the database systems which will really make you powerful and if you need we are there to help you.

As long as the course is on, the forum would be on. You can post in the forum beyond that also if you need help, please ask for it mail us and wish you all the very best with your course in your examination and the future course of your profession in life, all the very best.

**THIS BOOK
IS NOT FOR
SALE
NOR COMMERCIAL USE**



(044) 2257 5905/08



nptel.ac.in



swayam.gov.in