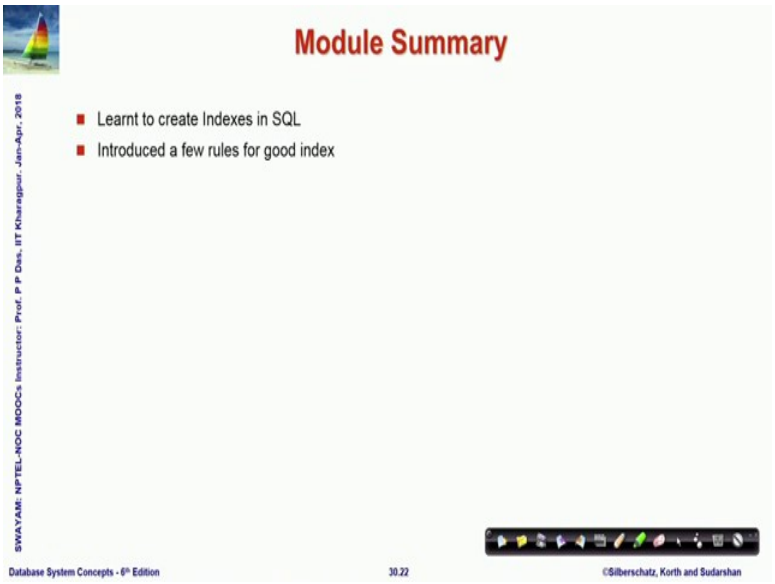
(Refer Slide Time: 23:26)



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.

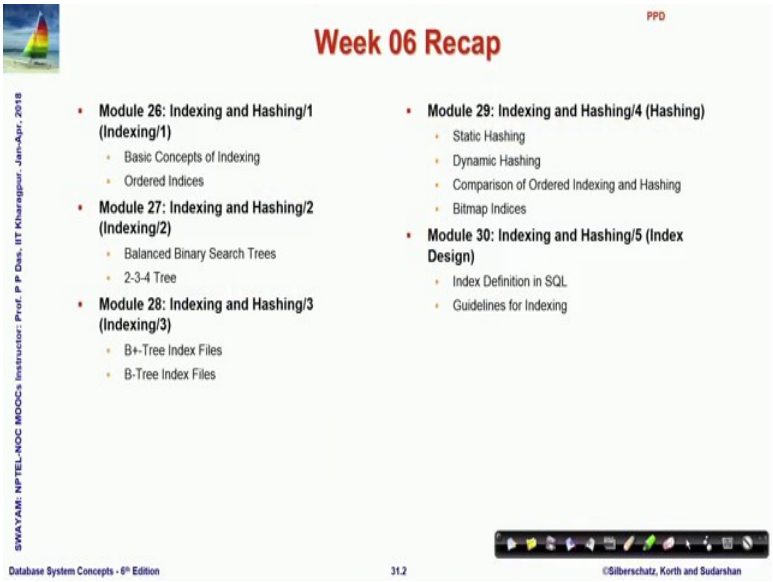
Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

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.

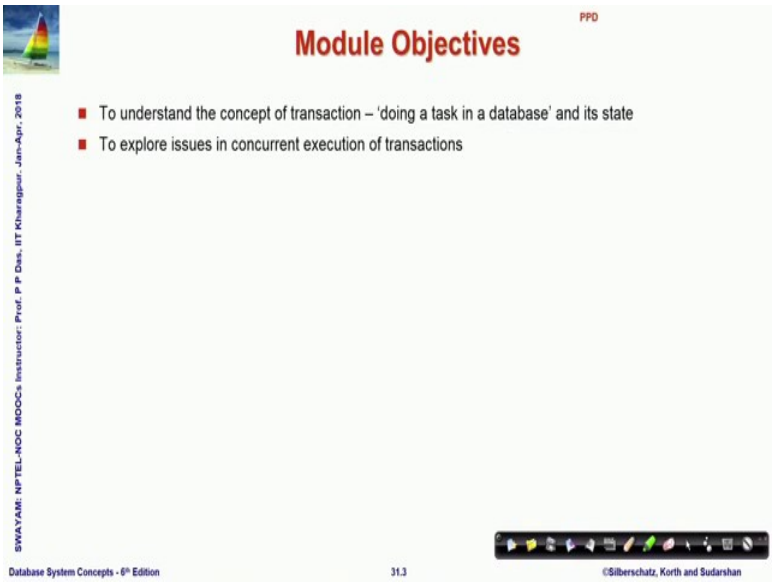
We have looked at how the data is actually physically stored, and how it can be efficiently accessed through different mechanisms in the storage. Now we will focus on the actual execution time. We will focus on what goes on when the data in a database system is accessed, it is read, locally modified 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)



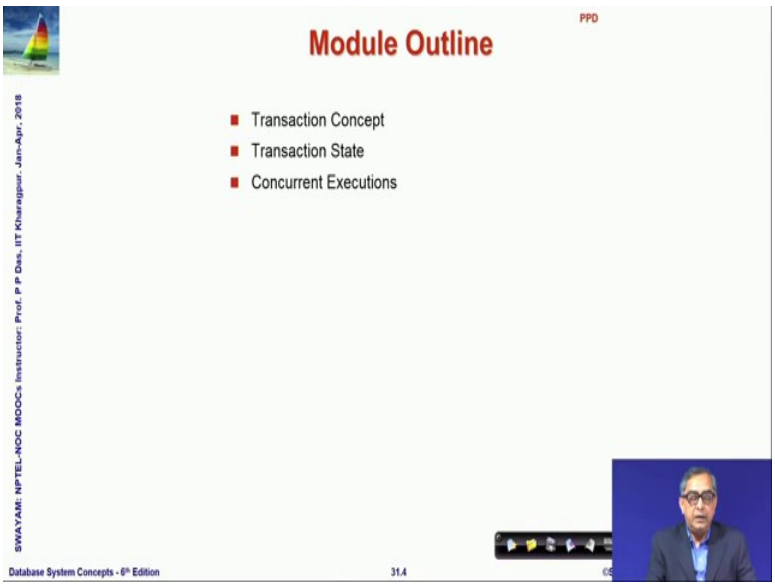
So, this is what we had done in the last week talking about index.

(Refer Slide Time: 01:46)



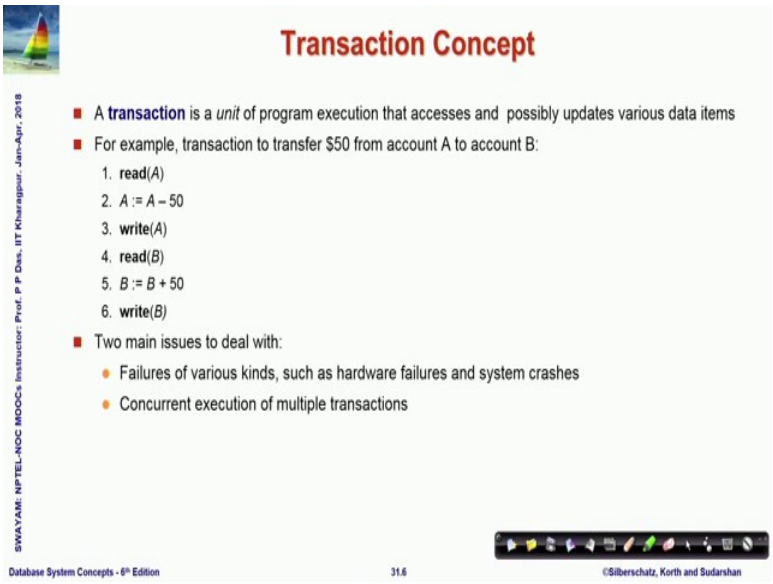
And we now start with the understanding of this concept of transaction, and we explore various issues related to concurrent execution of transactions. So, we will explain in more detail what this mean.

(Refer Slide Time: 02:03)



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

(Refer Slide Time: 02:13)



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 greets 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.

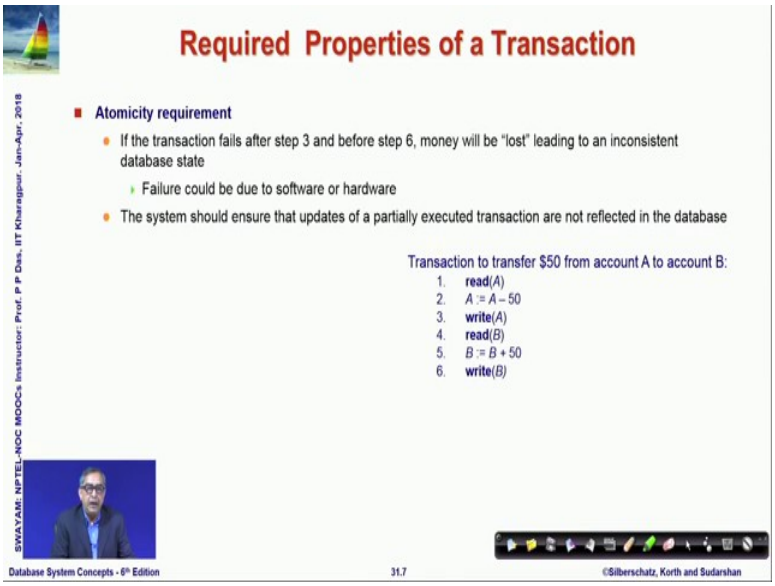
So, this whole sequence of 6 instructions is called a transaction. And as you can understand that to achieve our target of transferring 50 dollars from account A to account B, all these six instructions have to execute in this order so that we can get the desired results. Now the question is; so, this 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)

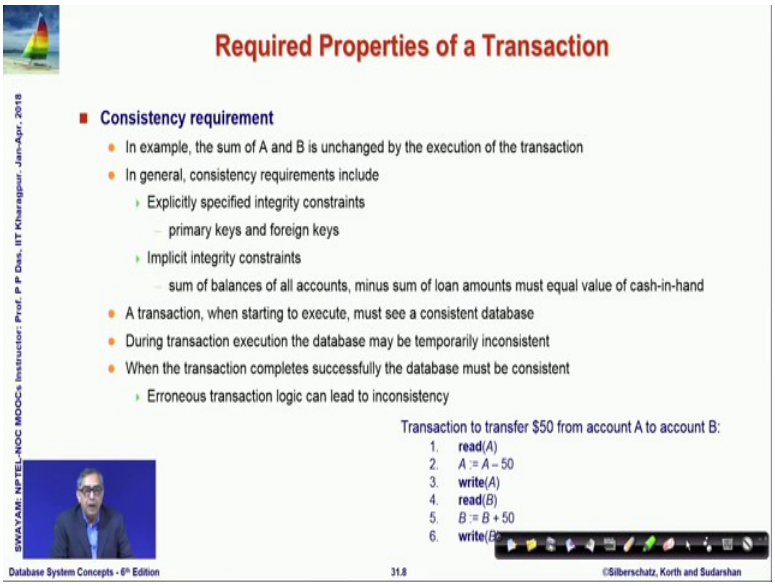


So, we first set the targets we put some required properties of a transaction. The first requirement is atomicity.

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 4^{th} 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)



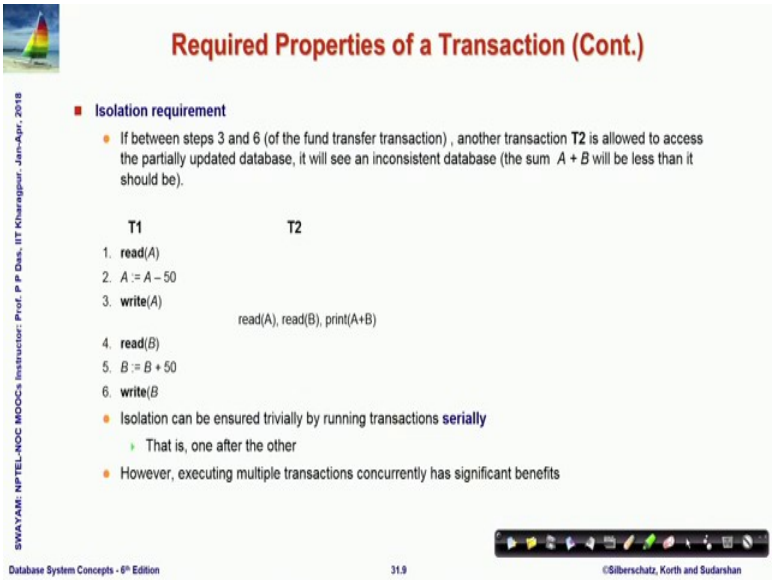
The second requirement is called consistency requirement. That is as the transactions are making changes to the database at, through these changes the integrity of the database the consistency of the values should not get affected.

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)



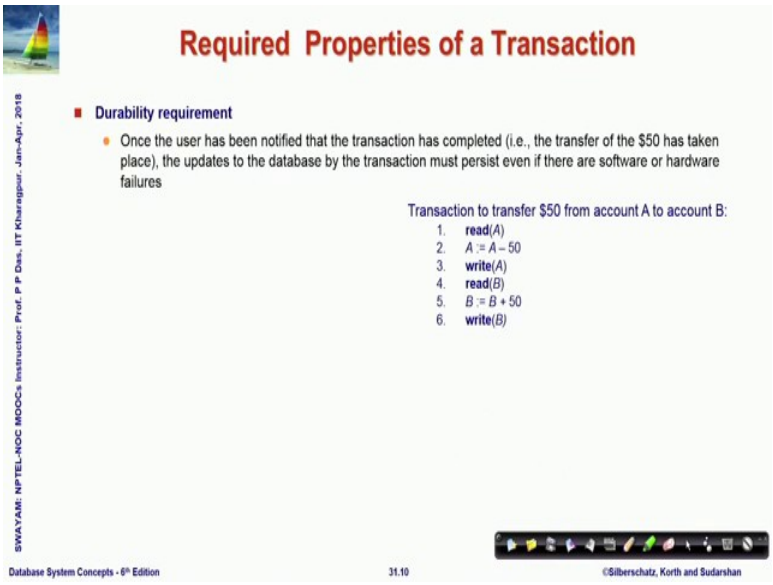
The third is again first look at the example the same on the left is a transaction T1 which is the transaction we have been talking of. And suppose there is another transaction T2 which happens concurrently. If it happens concurrently the transaction T2 has let us say 3 instructions read (A) read (B) 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 T 2 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.

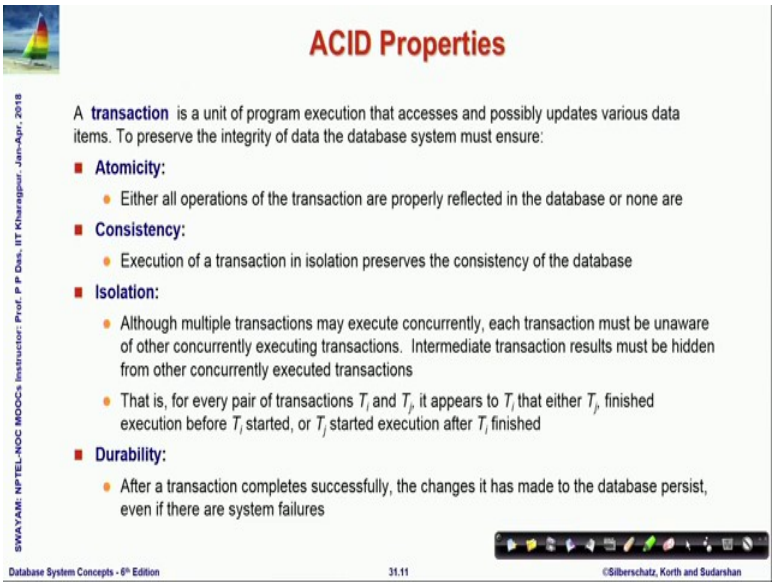
(Refer Slide Time: 12:24)



The 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)

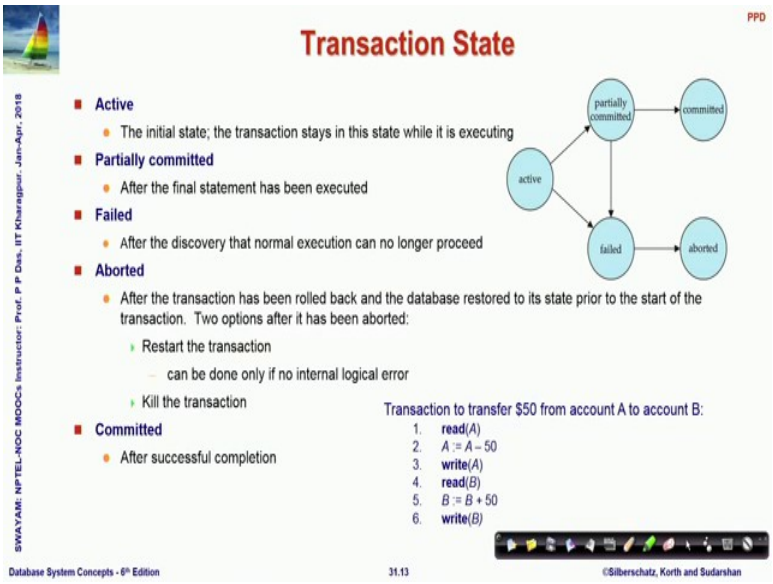


So, these 4 properties are together called the acid properties of a transaction system. So, acid means a for atomicity that either all operation of the transaction are properly reflected in the database or none of them are reflected consistency c for consistency execution of a transaction in isolation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

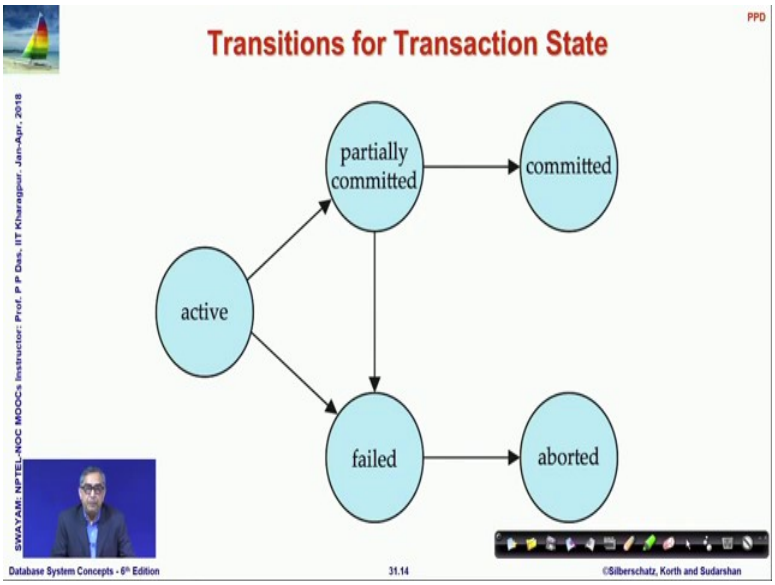
So, once it has executed the last treatment, last instruction that is instruction 6 here, it is in a state that is called partially committed. So, it has been able to successfully complete all the instructions. Or it might happen that during being in the active state, or being in the partially committed 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 4^{th} 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 likes 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.

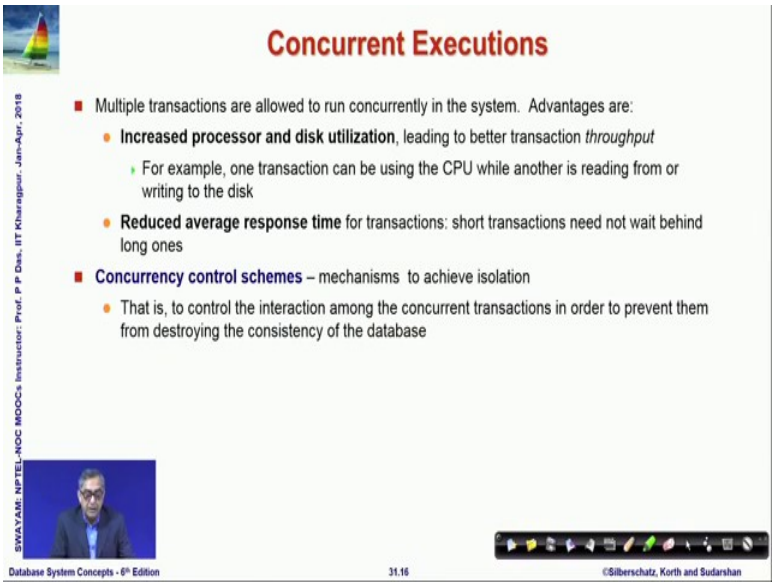
So, every transaction will go through this state at any point of time a transaction will be in one of the states, and depending on the status of execution it will continue to remain in that state or will change state.

(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)

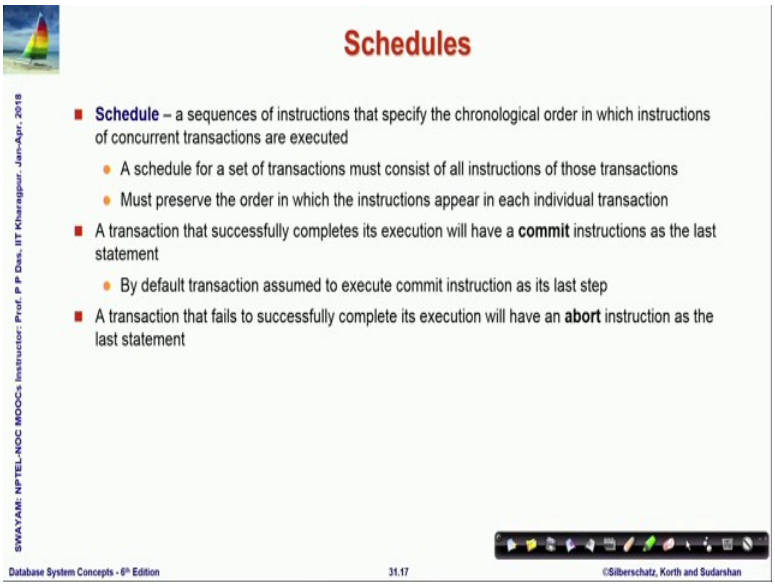


So, in the concurrent execution situation, what we have we have multiple transactions that run at the same time on the system. So, that will advantages it will increase throughput, it will increase processor and disk realization for example, when one transaction is doing some operations with on the CPU, 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.

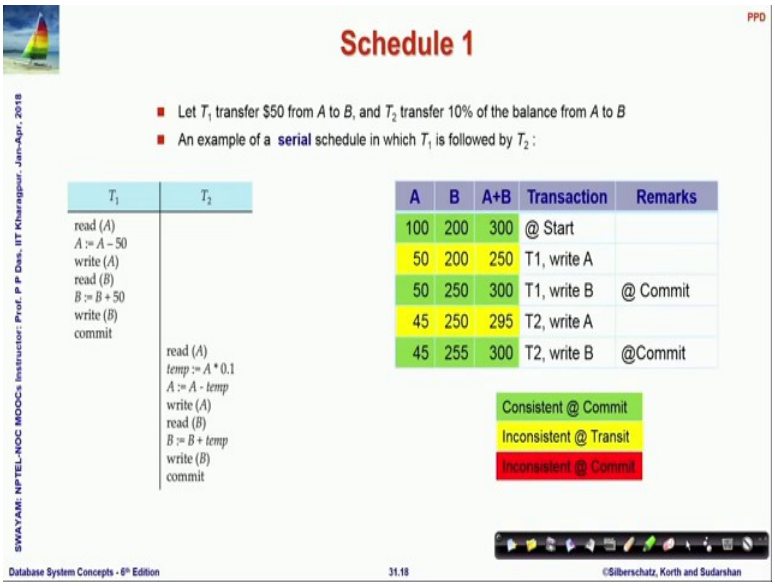
(Refer Slide Time: 18:50)



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 it is 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)



So, let us take an example so, again we are going back to the same example. So, we have two transactions T1 and T2. T 1 transfers 50 dollars from A to B as we have seen. And T 2 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 T 2 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.

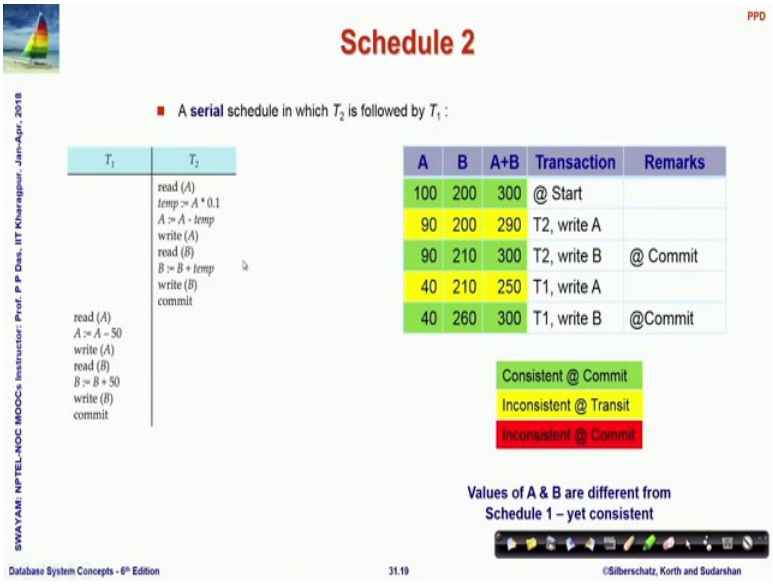
Then it reads B it reads 200 adds that 250 it writes B. You come to this point where after this write, when the commit is happening after the writing this B. Then T1 has actually completed, and 50 dollars has got transferred to account B, and the sum is again back to 300 so, consistency is preserved.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



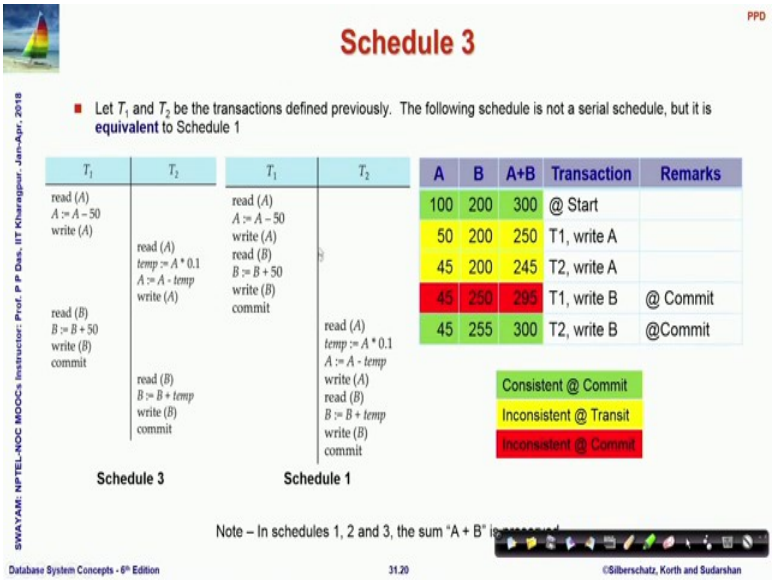
So, let us move on let us. So, this is a different schedule you can see, but this is also a serial schedule here what we have assumed that all instructions of T 2 are done first then all instructions of T 1. I am not going through the going through each step you can see what are the consistent, and the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



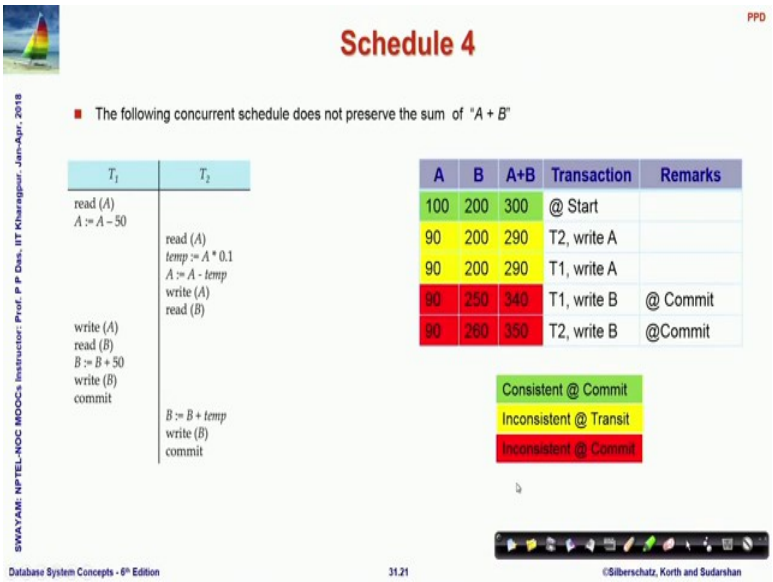
Now, take an interesting example, where schedule 3 where in here if you, if you look at carefully there are few instructions of T1 which are executed. And then in the temporal order few other instructs, few instructions of T2 are executed, then again T1 then again T2.

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 T1 writes A this is where you are 50 dollars has been debited. Then when T2 writes A subsequently, another 5 dollar is debited so, it becomes 45. So, then you have T1 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 T1 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 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)

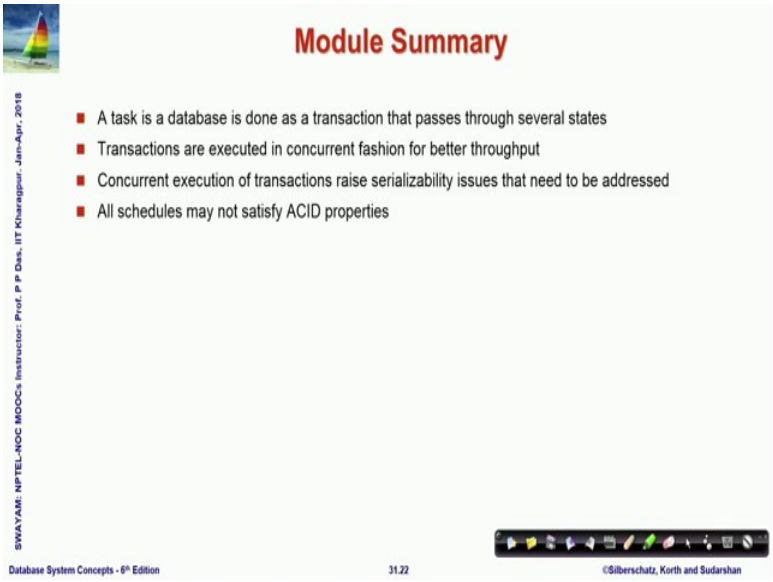


Now, look at again for those transactions look at a different interleaving, a different schedule, again T 1, T2 are involved. But you have now tried to interleave them in a in a different order. So, earlier the interleaving was done after T1 has done write A, here it is done after he has been the locally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



So, here in this module, we have understood the basic tasks that a data bit 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.

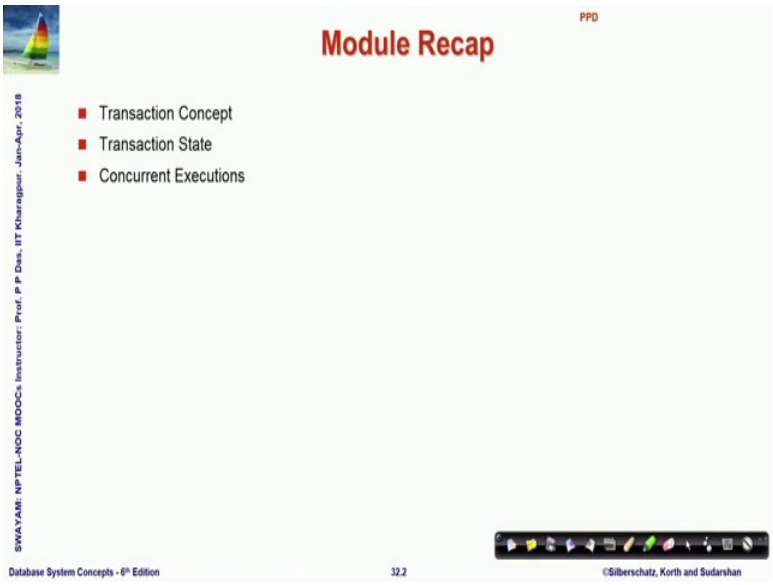
And those need to be addressed that certainly inconsistent database is certainly never acceptable. And so that is the basic problem that we have identified which we will have to address in the coming modules.

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

Lecture – 32
Transactions/2: Serializability

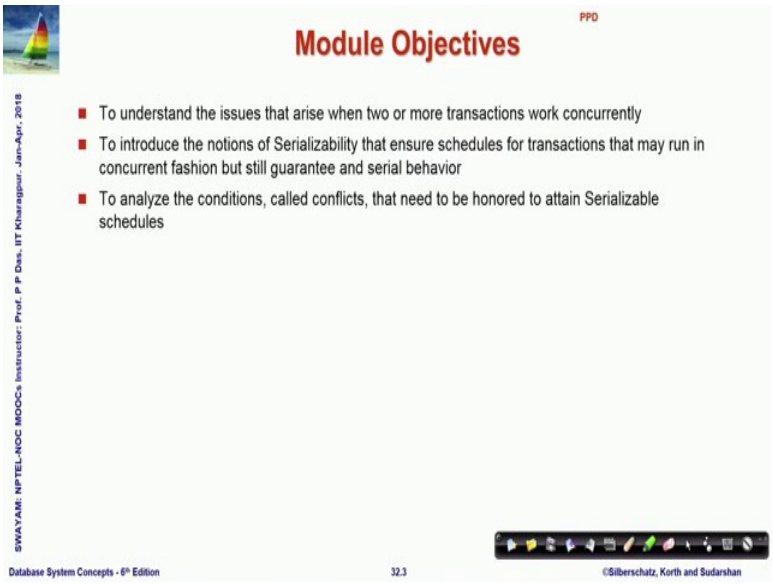
Welcome to module 32 of Database Management Systems, from the last module we are discussing about transactions and transaction management.

(Refer Slide Time: 00:27)



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

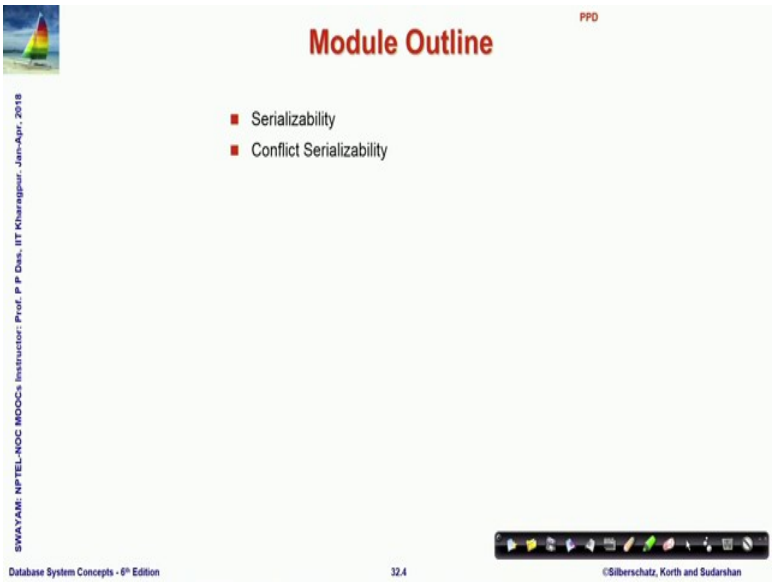
(Refer Slide Time: 00:35)



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.

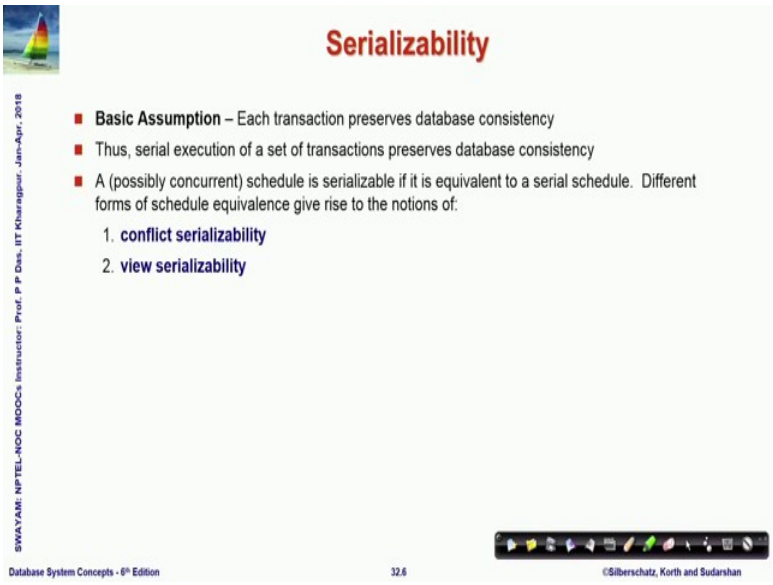
So, we want to introduce the very basic concept of making sure that such concurrent execution schedules are acceptable, and those are the notions of serializability. And we will analyze different conditions called conflicts that need to be honored to attend the serializability of the schedules.

(Refer Slide Time: 01:17)



So, serializability is the main topic to discuss.

(Refer Slide Time: 01:21)

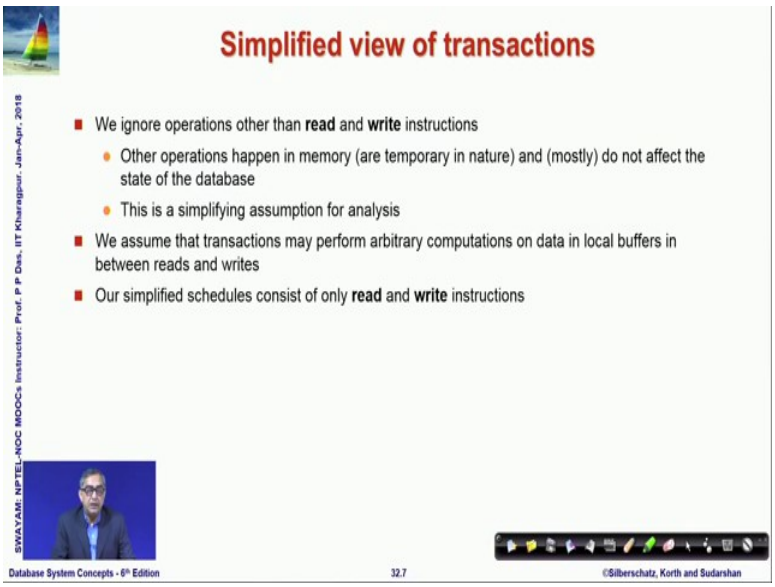


So, to understand serializability we make a basic assumption, we make an assumption that every transaction by itself preserves the database consistency. That is, it starts in a consistent state of the database. And through the execution of it is instructions in the order given it leaves the database 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)

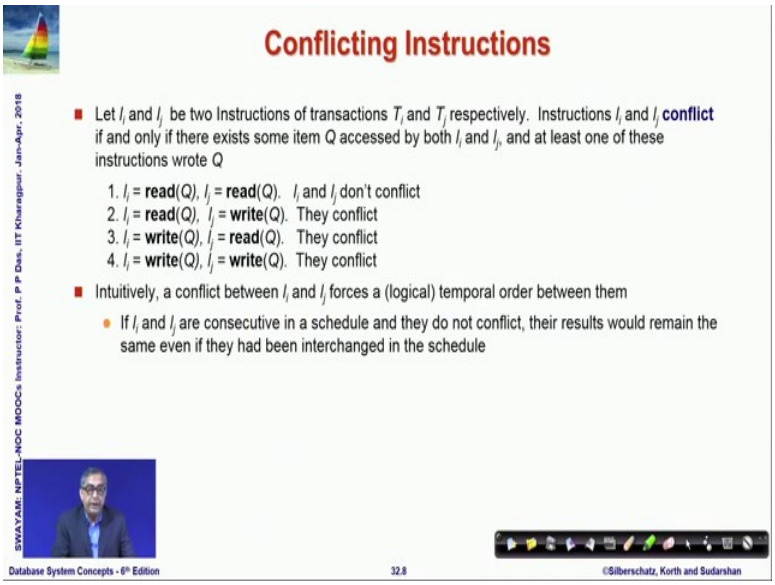


Now, we make now a transaction may have all varied kinds of instructions, but we make an assumption that we will ignore anything other than any instruction other than the read and write instruction. Because other operations like we saw an operation where an account is debited by 50 or account is credited. 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)

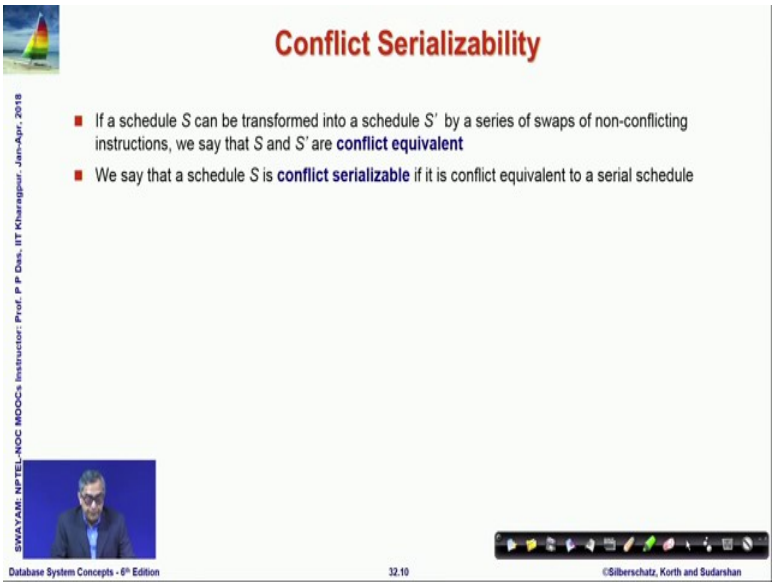


Now, we say that suppose I_i and I_j, 2 instructions for belonging to transaction T_i and transaction T_j. So, there are two transactions T_i and T_j, T_i has an instruction I_i, T_j has an instruction Ij and we say that I_i ad I_j this instruction will conflict, if and only if there is some item Q, 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)

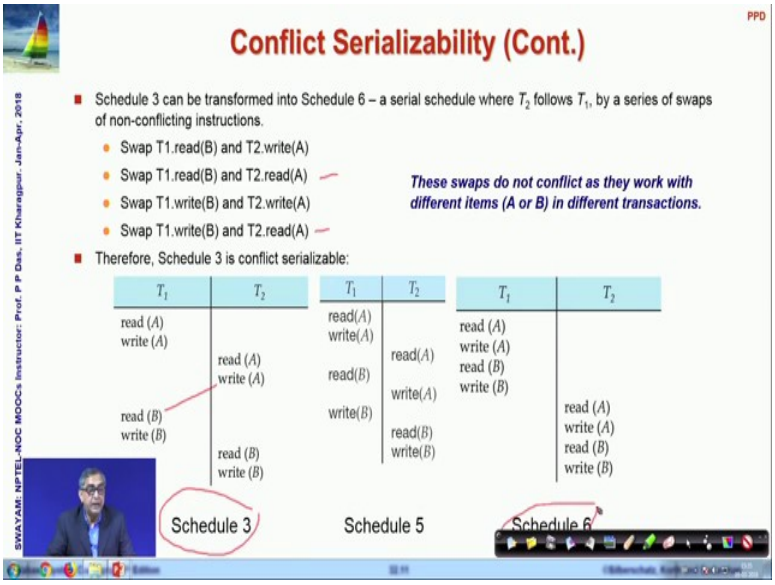


So, that gives rise to the notion of conflict serializability. So, we say if a schedule S can be transformed into another schedule S' by a series of swaps of non-conflicting instructions, then S and S' that conflict equivalent. So, what are you saying? That we have two one schedule S, and we will swap 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)



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.

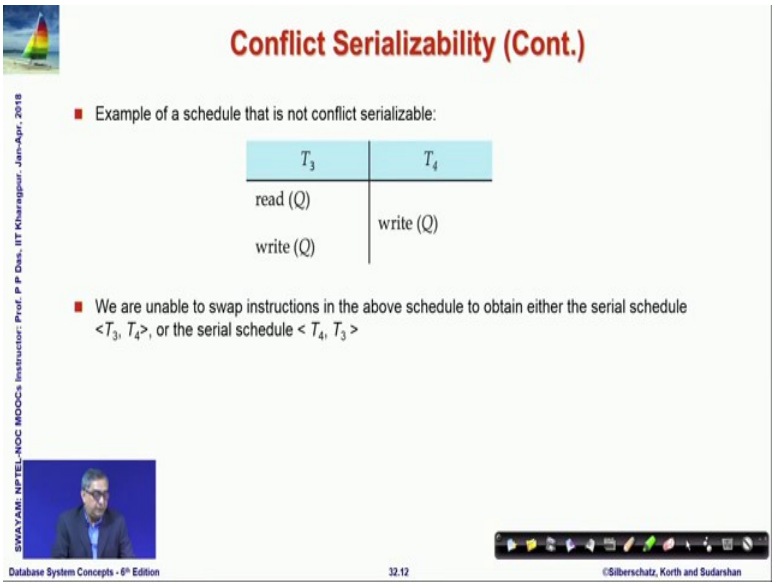
So, since this can be done, we will say it is conflict serializable schedule 3 is conflict serializable and just to see how that happens. So, you start here let me erase this marks and start here. So, here if I look into these two instructions, which are the consecutive instructions in schedule 3 I can 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)

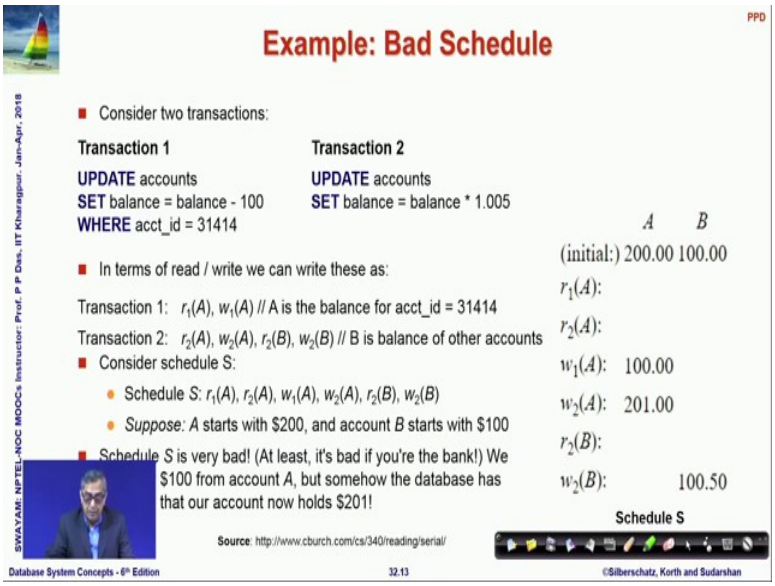


Just as very simple example suppose you had two transactions T_3 and T_4, and you have this situation. Now is it conflict serializable it is not. Because to make it conflict serializable. I need to either swap write(Q) of T_3 with write(Q) of T_4 which is not possible because these are conflicting instructions,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

The other option is I could swap read(Q) in T_3 and write(Q) in T_4, 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 T_3 T_4 or to T_4 T_3. 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 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)



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 r 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 if you look at here on the write this is schedule S.

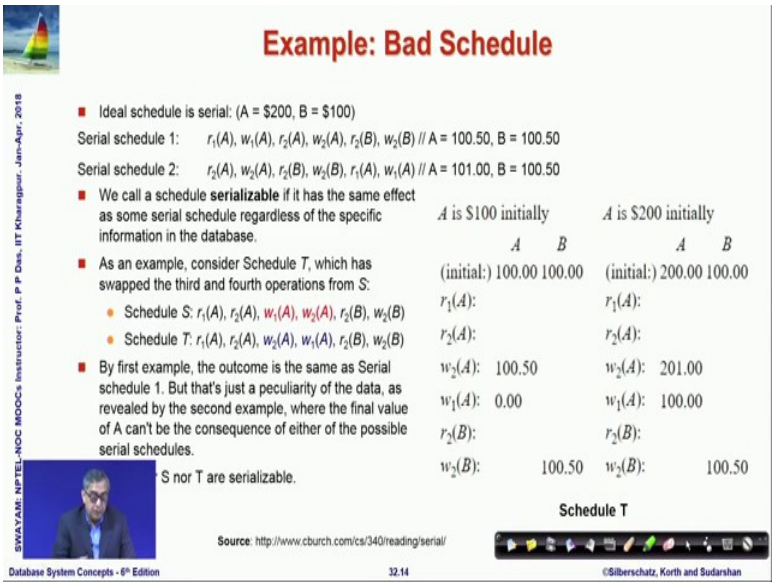
So, in the write we are saying, that let us say that A starts with 200 dollar, and B at the beginning is 100 dollar. So, what will happen you will read? You will read here this is the first thing $r_1(A)$. So, 200 is read then $r_2(A)$. So, what happens if $r_2(A)$ is read Again 200 is read. And then 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, w 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)



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

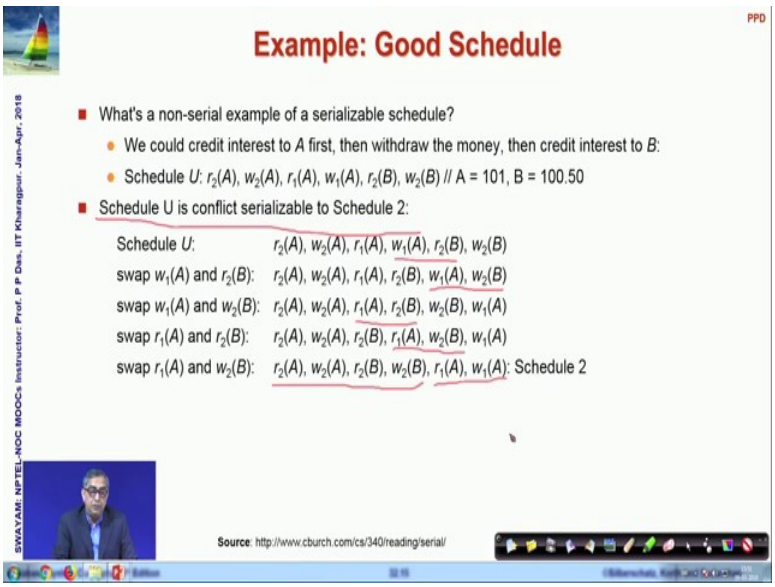
happens so, the interest is credit 100.5. And then what has happened? w_2 after that $w_1(A)$ so, whatever was written here is debited and written back. So, whatever was read there is 100 dollar. So, you debit 100 dollar it becomes 0.

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 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 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)

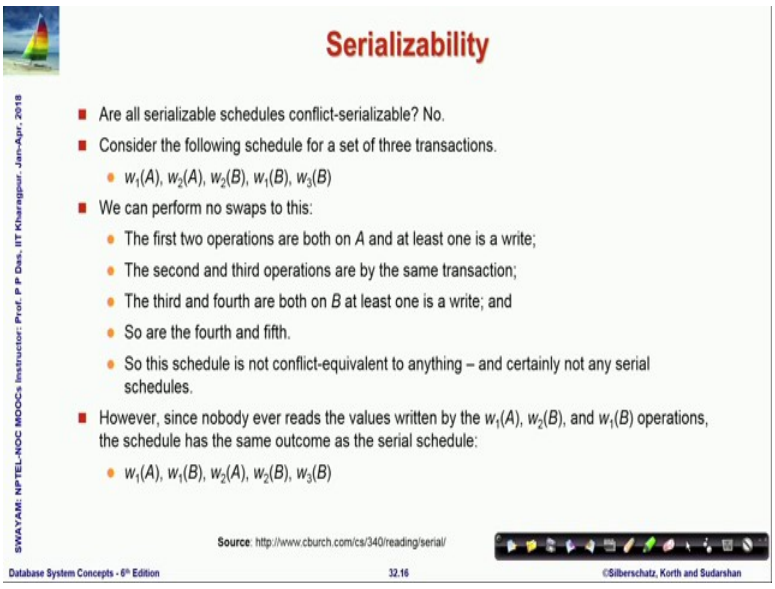


So, neither S naught T are serializable, yet another schedule U this again. So, you can see that transaction 1 is happening instruction of transaction 1 is happening somewhere in the middle. With transaction 2 and this is what you get. So, if you if you look back as to earlier case. You will find that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

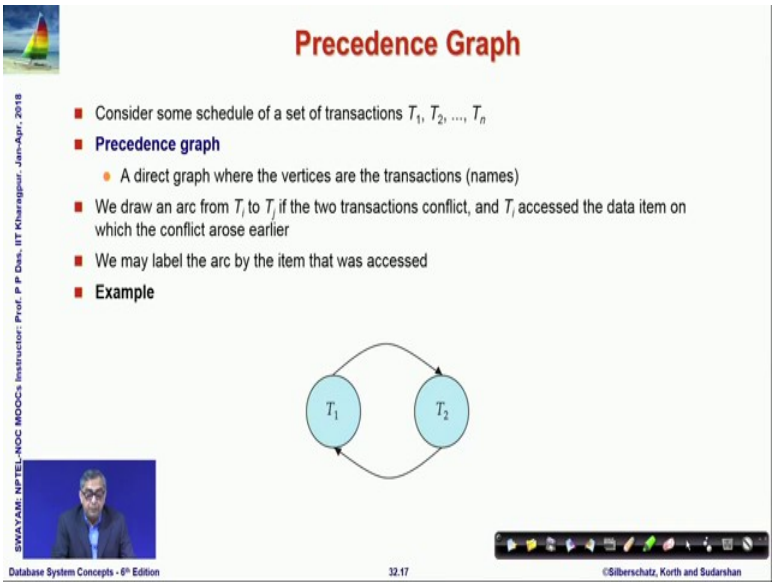
(Refer Slide Time: 26:09)



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.

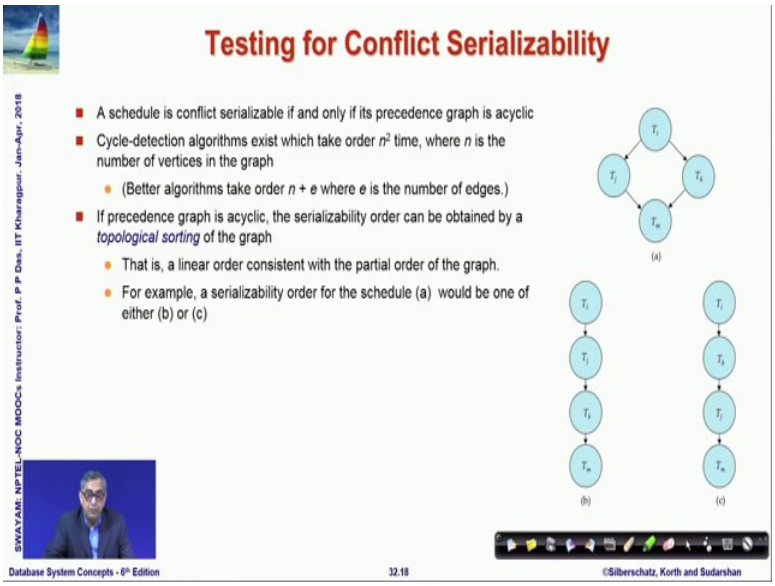
(Refer Slide Time: 27:36)



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

conflicting. So, if T_i T_j conflict there will edge between that. 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 art 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.

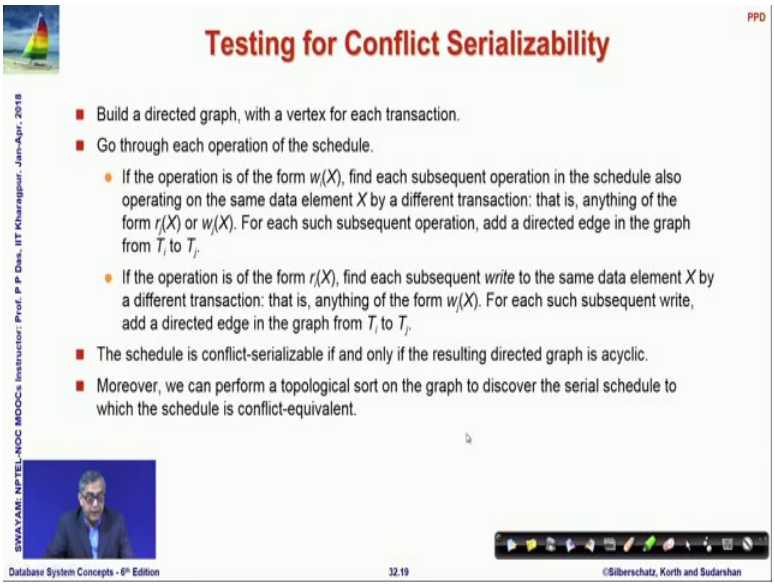
(Refer Slide Time: 28:47)



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)

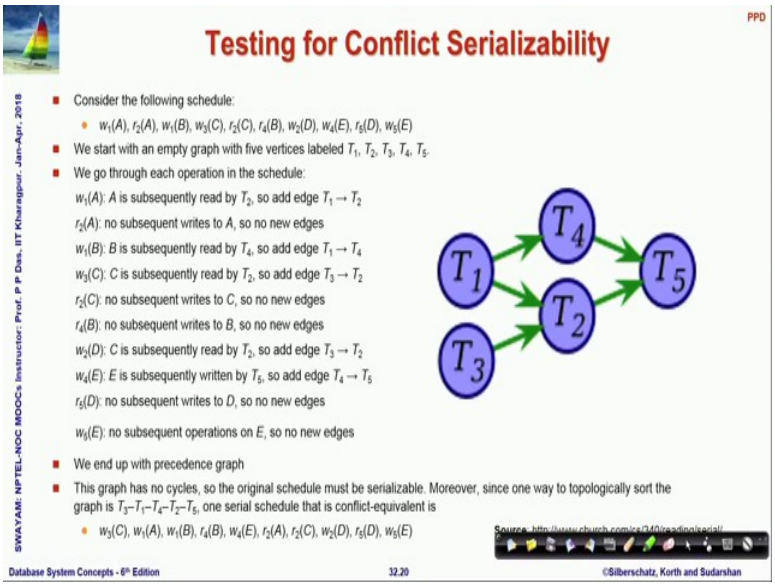


So, to test for conflict serializability; the steps will be build the directed graph. Then go through each operation of shall, you look at each operation read or write. If the operation is a write, then find so, if it is $w_i(X)$, then find what is happening with data this data element X in different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



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 A is here read by T 2.

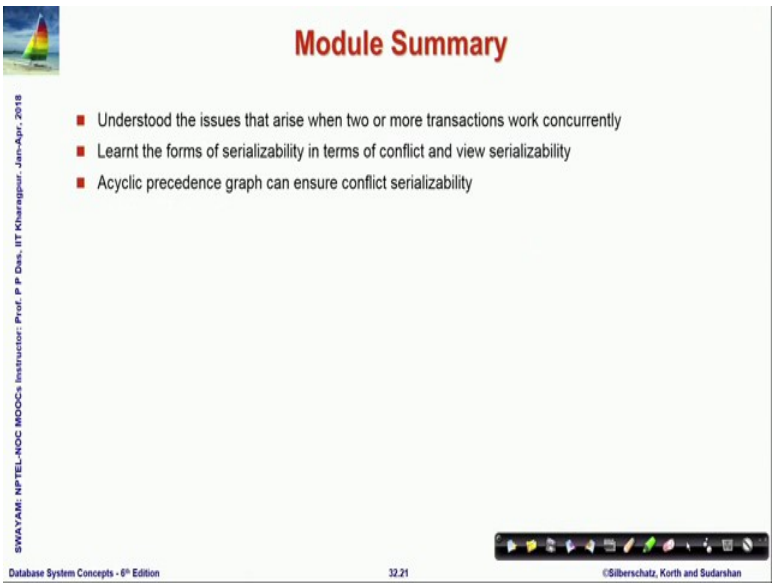
So, there is a conflict. So, you will add an edge T_1 T_2 so, this edge gets added. Then is to have $r_2(A)$, and you find look for A, A, A, A, A, A there is no A so, there is no subsequent write. So, there is no new edge then you have w 1 B, w 1 B and if you look for you have $r_4(B)$; so r_4 that this 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 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)



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

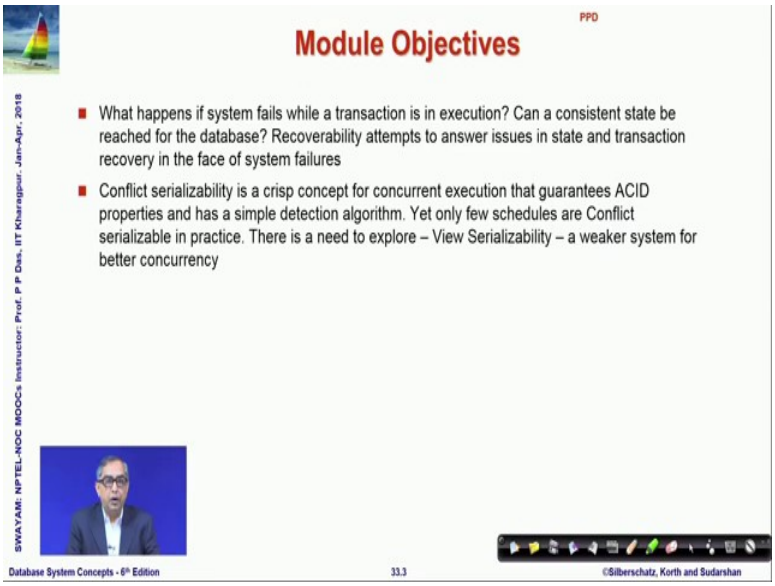
Lecture – 33
Transactions/3 : Recoverability

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)

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

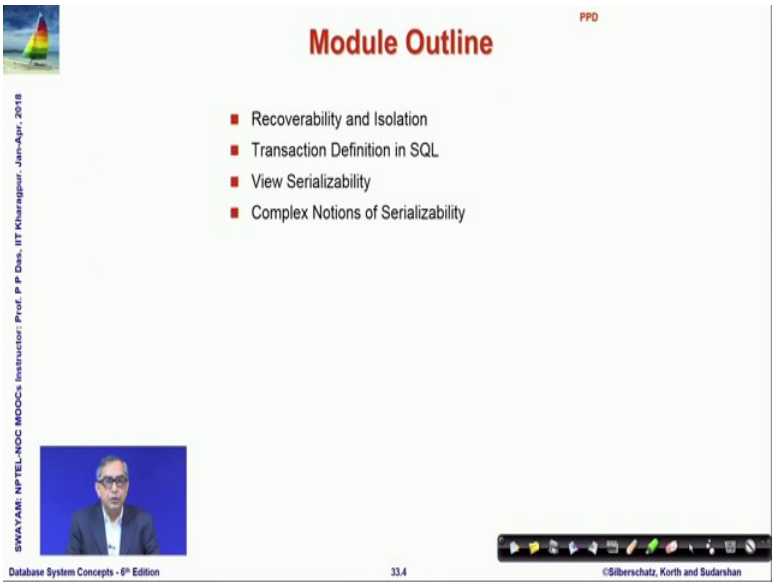
(Refer Slide Time: 00:50)



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.

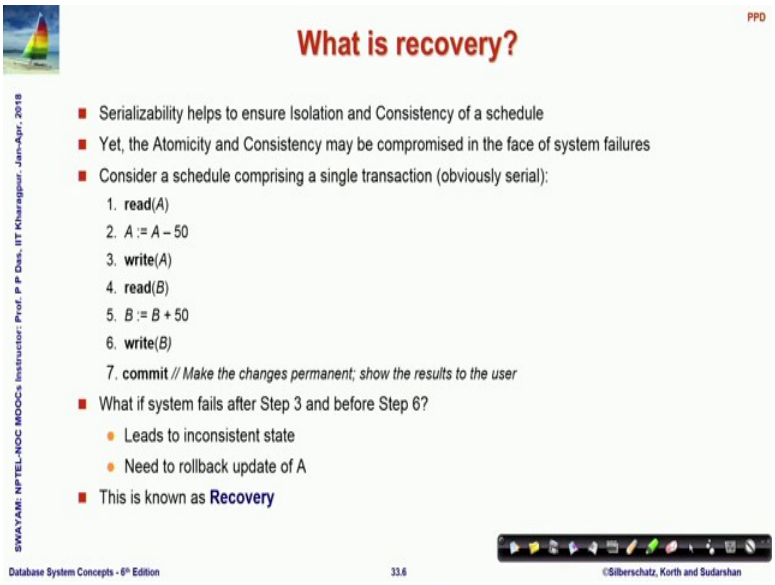
We would also look at that going forward from conflict serializability, what are the other notions of serializability, that can be used to serialize transactions and we will look at a weaker definition of serializability known as view serializability, which can serialize more schedules than what conflict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

(Refer Slide Time: 01:51)



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

(Refer Slide Time: 01:57)



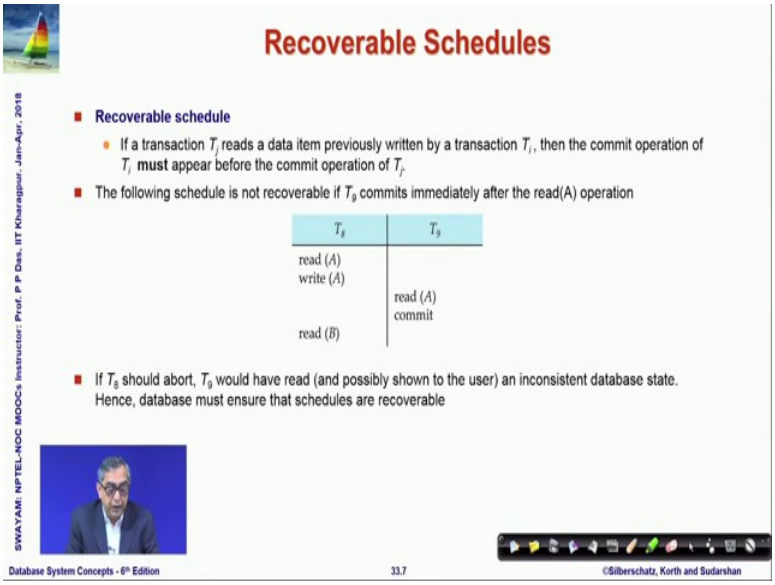
So, what we have done is we have seen the serializability help us, if we think in terms of the acid properties that we started by defining as the desirable properties of the transactions, we have seen that the serializability significantly helps us to achieve isolation and consistency of a schedule, 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 between 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)



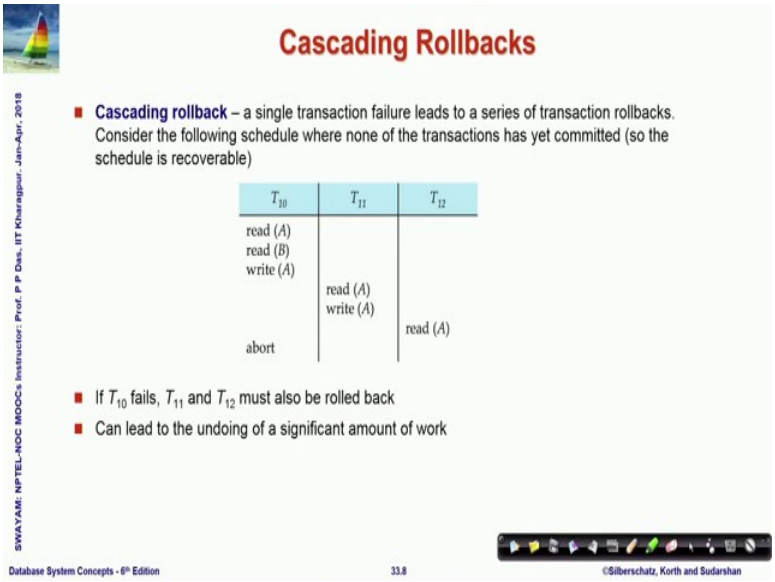
So, we say that a so, let us define a schedule to be recoverable if a transaction T_j reads A data previously written by a transaction T_i, then the commit operation of T_i must appear before the commit operation of T_j, if that happens then that is the earlier transaction which has written the data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

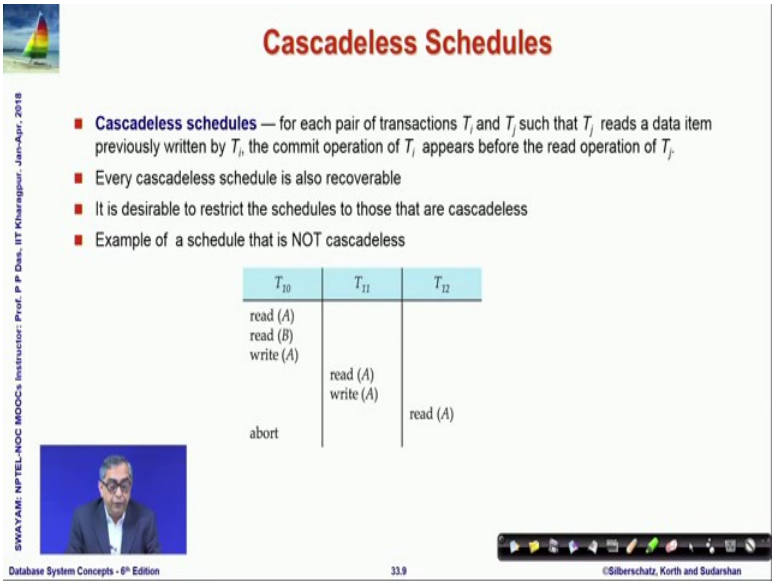


So, this is an example of a schedule which is not recoverable. Now let us also observe that a single transaction failure not only means that one transaction needs to be rolled back, but it could have a cascading effect, that is a series of transaction may require a rollback. So, here is an example of 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)

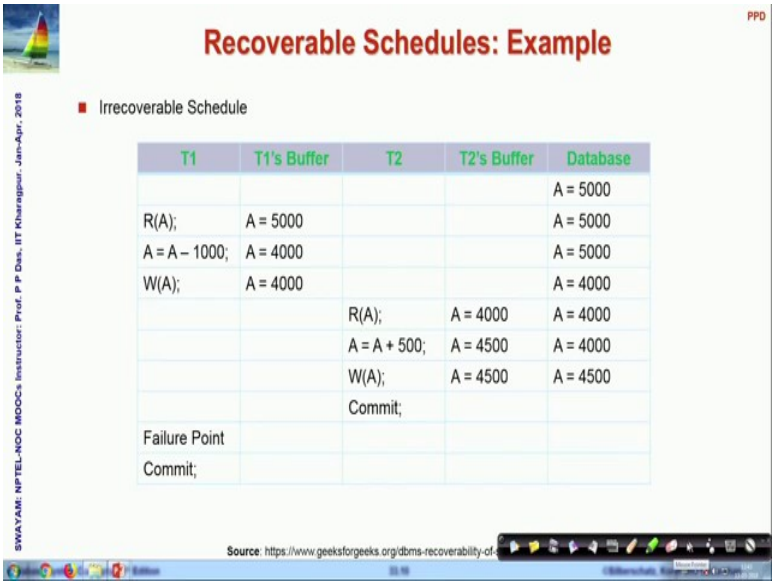


So, what we would prefer is if we could have schedules where such cascading roll back is not required. So, and there is a there is a condition through which you can achieve that. So, if we have a pair of transaction T_i and T_j. So, that T_j reads A data item previously written by T_i, then the commit 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 cascadable 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)



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 in the database the initial value of a is 5000.

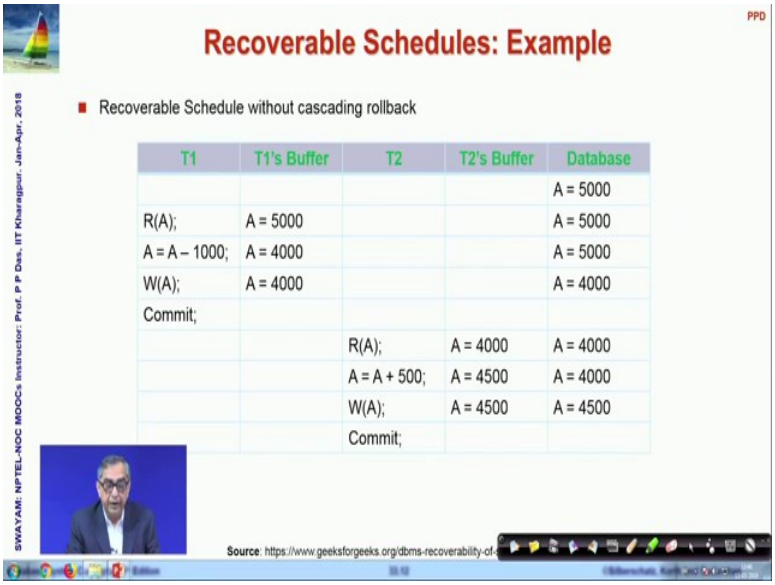
So, what will happen is read here and this value is a different A this is in the buffer or the memory of T_1 transaction, where A becomes 5000, then you subtract 1000 and then you write back the moment you write back in the database in between the value in the database is not changing, it is only that 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)



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.

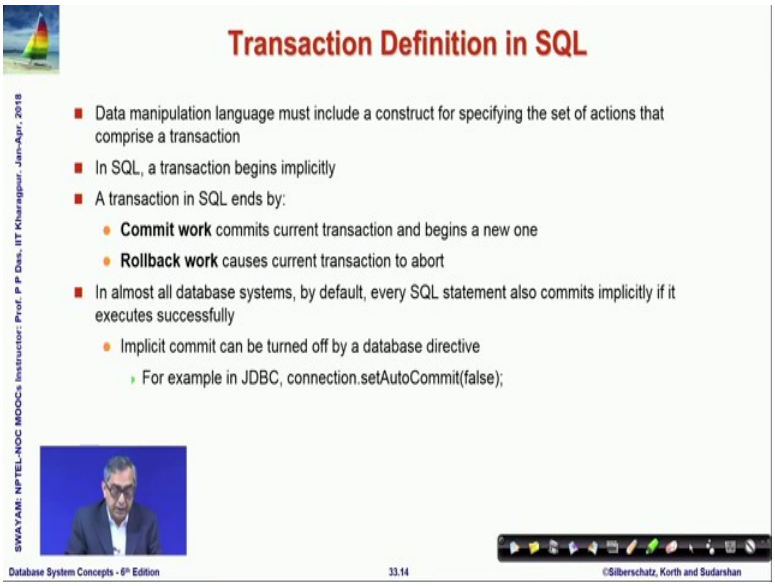
So, the transaction it is reading from it is actually committing the changes after that source transaction has committed. So, that satisfies the condition of recoverable schedule. So, you are able to recover, but it still required the cascading because T 1 had read A value in here of A which was not 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 T_2 reads that only after that commit has happened, earlier it was reading before that commit has happened. So, once T_2 reads it after this commit. So, if there is some there is some requirement of if there is some situation of rollback, then only T_1 needs to be rolled back and T_2 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.

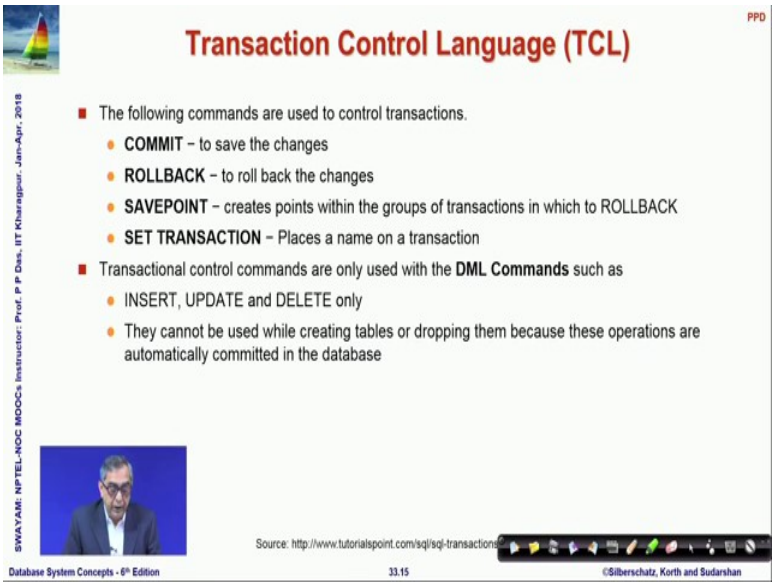
(Refer Slide Time: 14:12)



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.

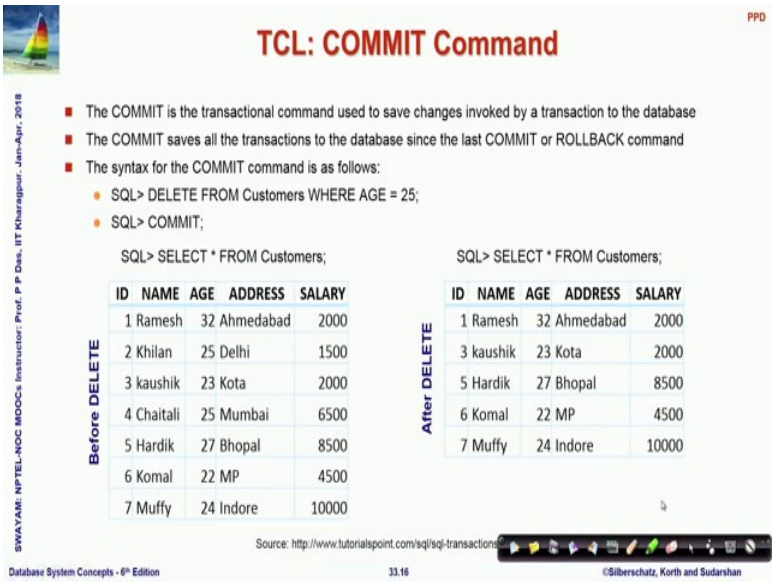
(Refer Slide Time: 15:22)



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.

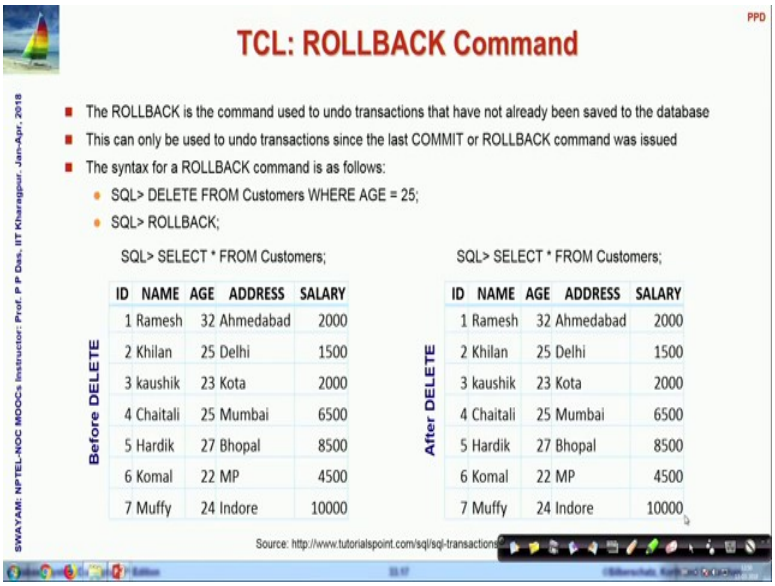
(Refer Slide Time: 16:09)



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.

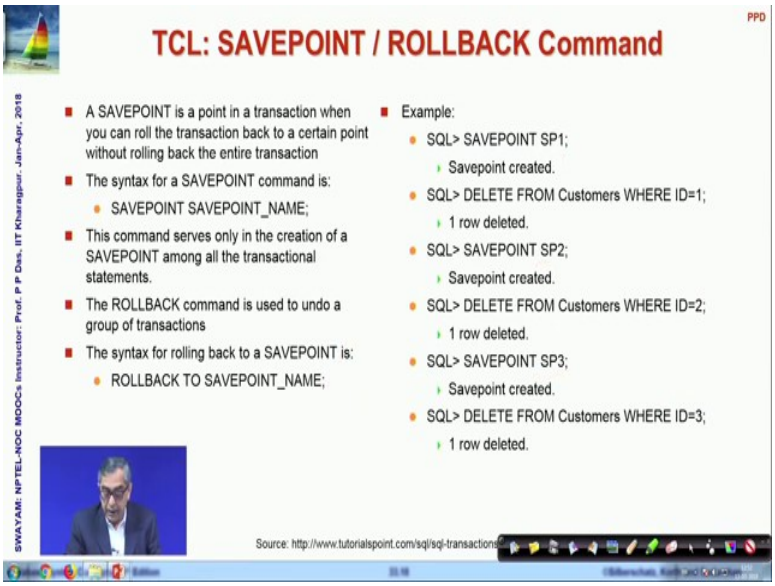
(Refer Slide Time: 17:11)



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)

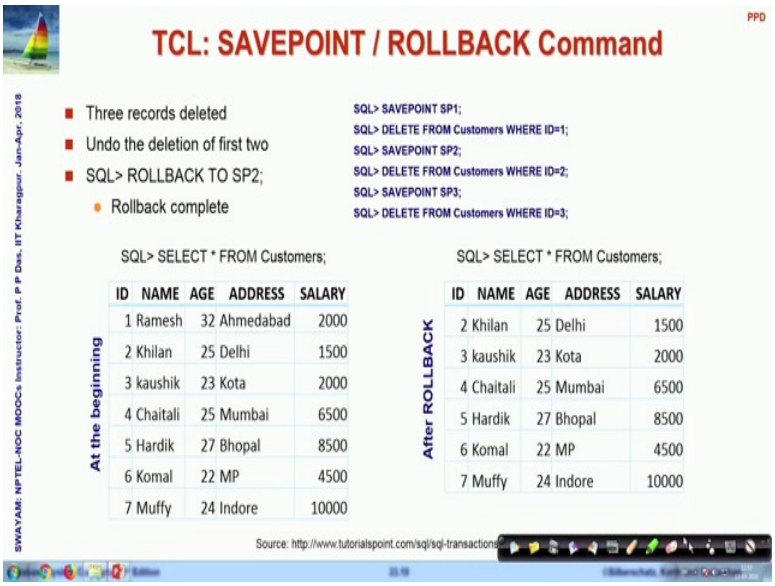


Now, you can a transactions often could be long. So, within the transaction you may want to mark certain points. So, that in case you roll back or you need to roll back, you can roll back to that particular point and those points are in the transaction are known as the **SAVEPOINT**. So, this is the 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 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.

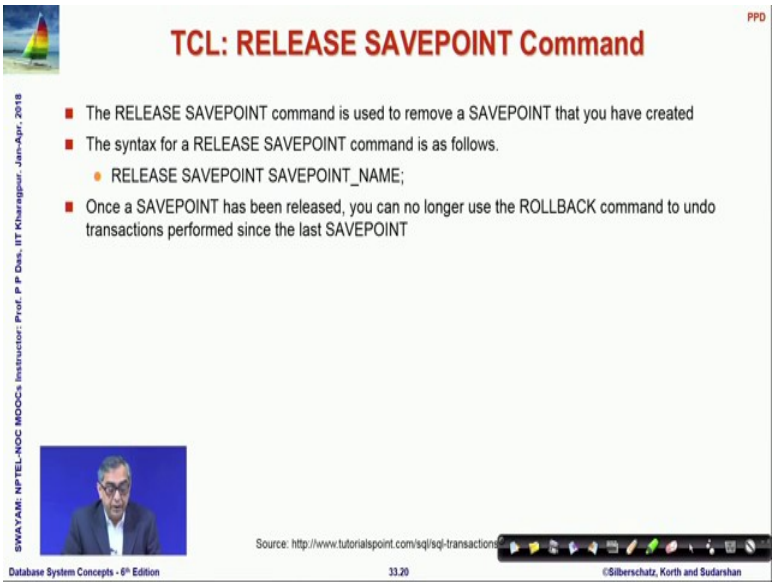
(Refer Slide Time: 20:14)



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.

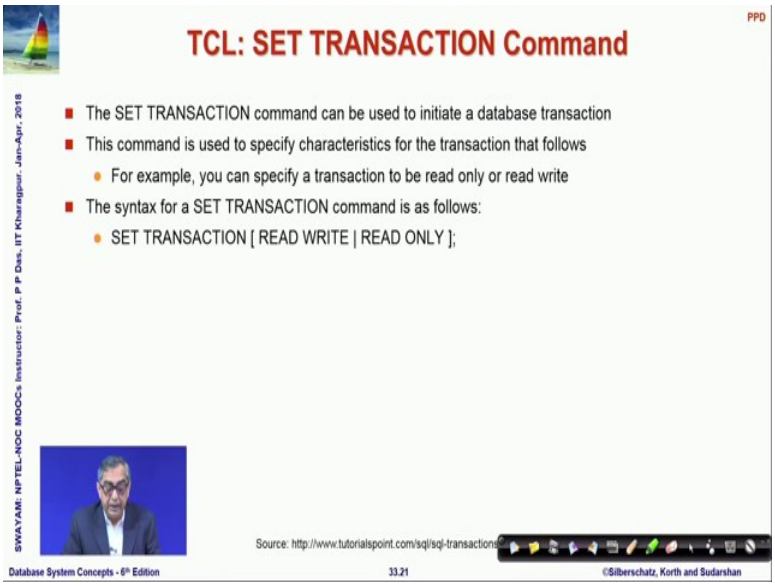
So, then when I undo the deletion of the last 2 records, then the what I see is the records which are marked as ID 2 and ID 3, which were done after SP 2 was marked which were deleted after SP 2 are marked, they are back into the table whereas, the deletion of SP 1 is still in effect and therefore, deletion 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)



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)

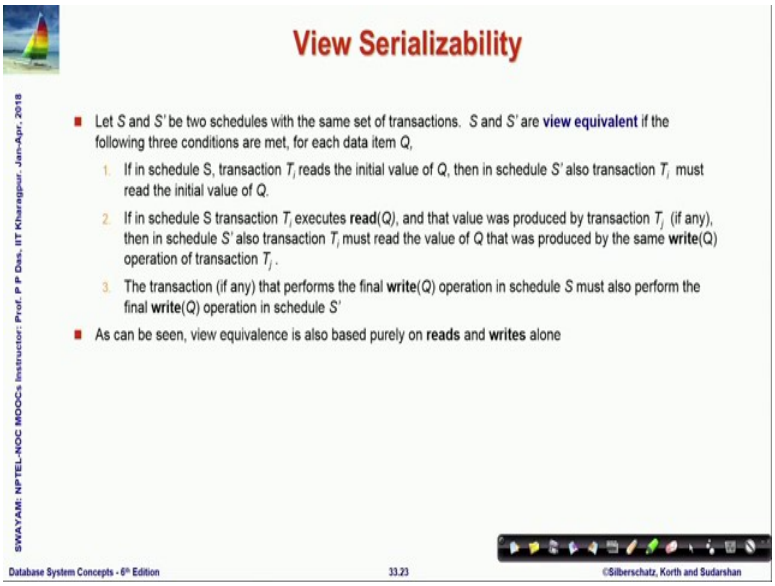


You can use **SET TRANSACTION** command to initiate a database transaction also and, it is typically used to specify the characteristics of the transaction, particularly if you want to say whether a transaction is a read only transaction or a read write transaction,

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)



So, in terms of view serializability we again define what is known as when are 2 tran 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.

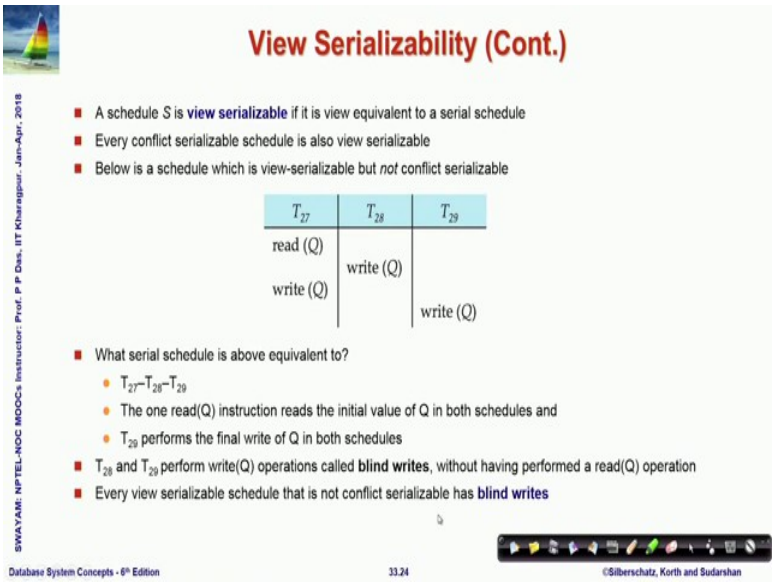
Similarly, the third condition says that the final write that is done, final value that it writes every transaction writes in both the schedules must be the same the same writes should operate. And the second conditions is a read write pair that every transaction when it performs a read on the data item, 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)



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.

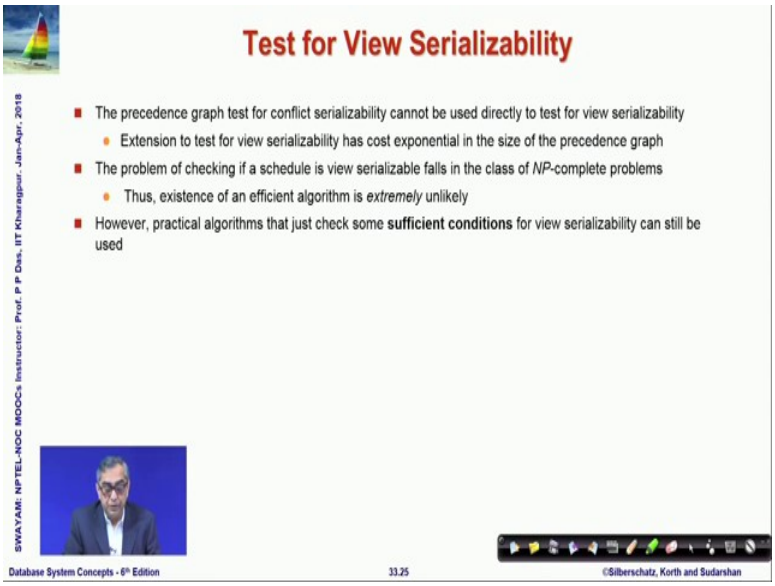
So, here is a schedule which is view serializable, but it is not conflict serializable, you know this is not conflict serializable because, certainly you cannot make it into a serial schedule make it equivalent to a serial schedule because, you cannot move this right Q above the right Q of T_{28} or of T_{29}, 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)



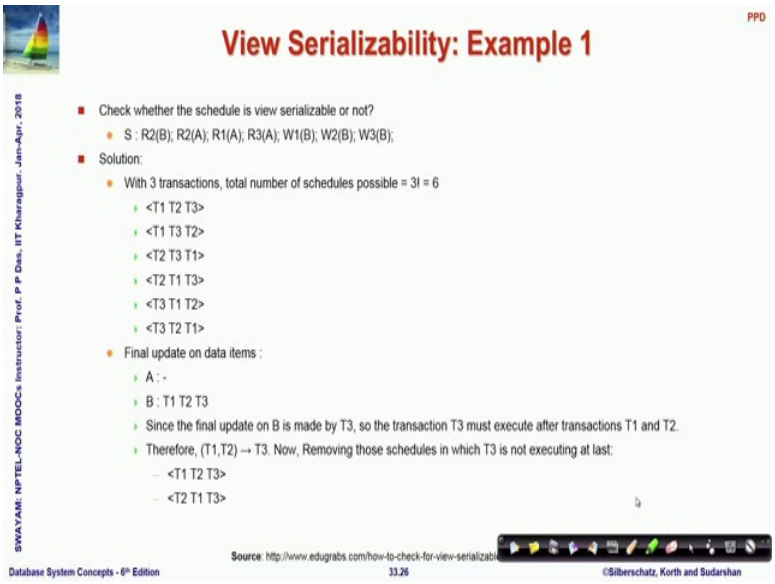
Now the question is similar to conflict serializability, where we saw that it schedule can be conflict serializable, if it is corresponding precedence graph is a cyclic. So, we would

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 in efficient algorithm for.

It 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)



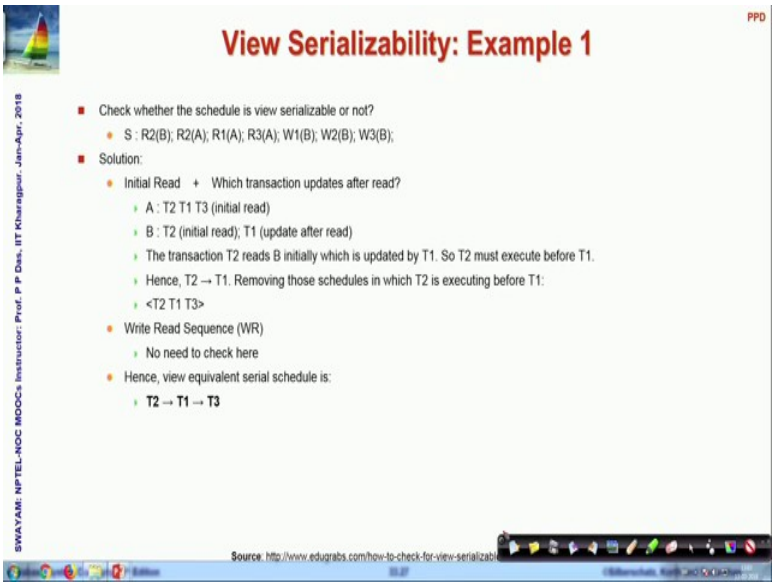
So, using view serializability have certain problems. So, here I have worked out a longer problem in terms of the view serializability to check that. So, it is kind of a brute force algorithm. So, if you see this is the schedule given there are two data items A and B and

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)



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.

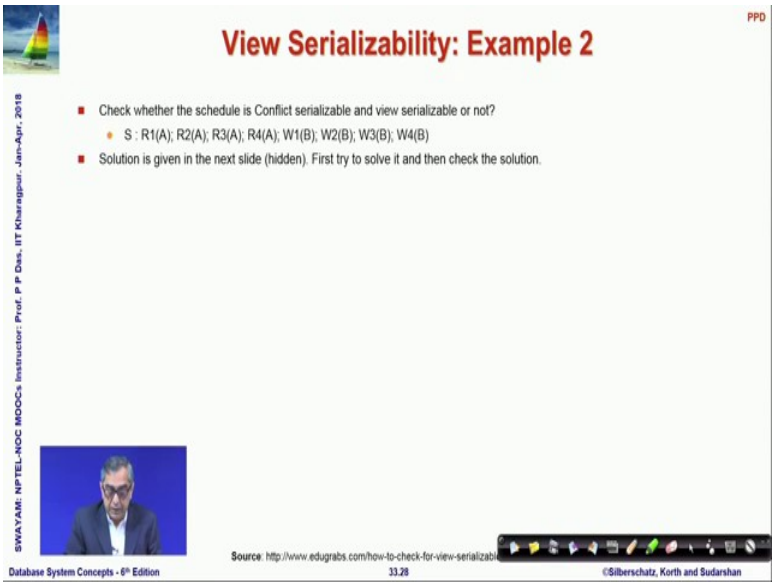
So, condition one checks that they must read the same value in both the schedule. So, we see that these are the reads that are happening on A. So, we see that on a there are reads happening, I am sorry this is these are the three that is reading A. So, it happens in the order of T_2 T_3 T_1 and T_3.

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 3.

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.

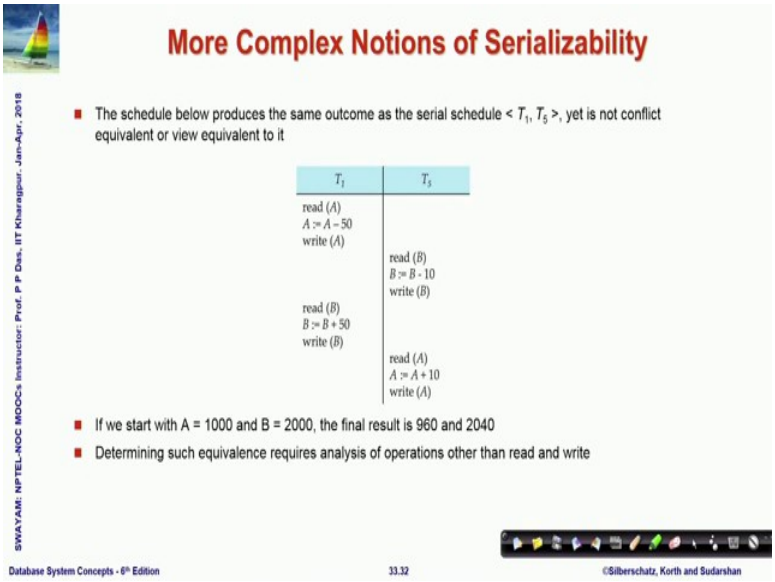
(Refer Slide Time: 31:45)



There is another example given here, where there is 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)



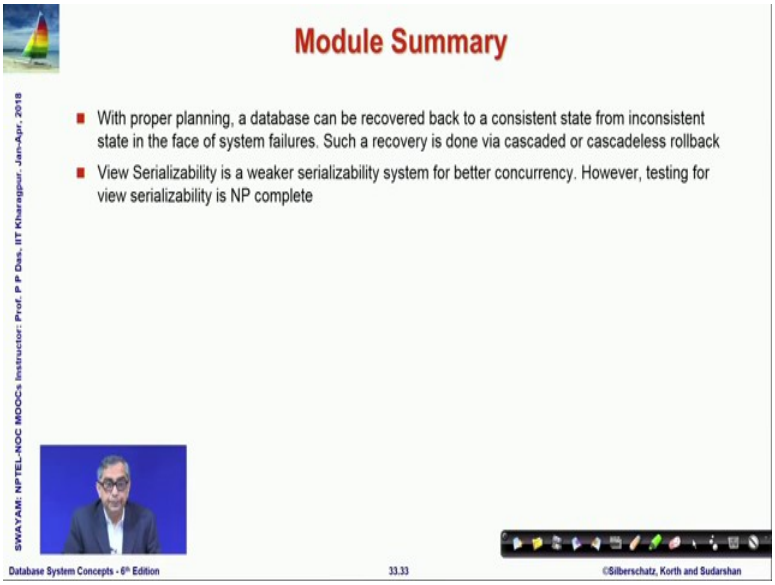
There are different other complex motions of serializability also for example, if you look at this particular schedule this actually is a serializable schedule, this is the effect that it produces will be same as the serial schedule of T1 T5, but if you go through the definitions of conflict equivalence 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)



So, we have shown that with we have shown here that with proper planning, a database can be recovered back to a consistent state from an inconsistent state, in case of system failure.

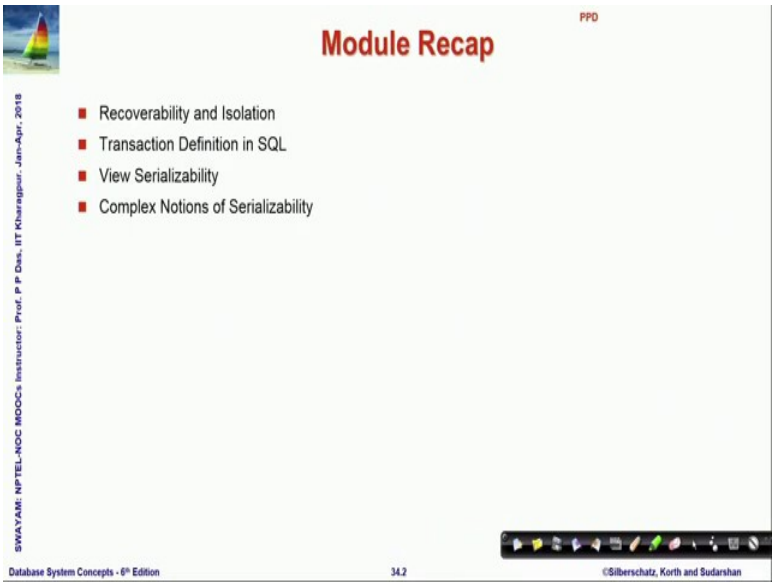
And this such a recovery can be through cascaded or cascadeless rollback and, we have also introduced a simpler model of serializability in terms of the view serializable, but testing for view serializability is np complete. So, as an effective algorithm it is not that powerful.

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

Lecture – 34
Concurrency Control /1

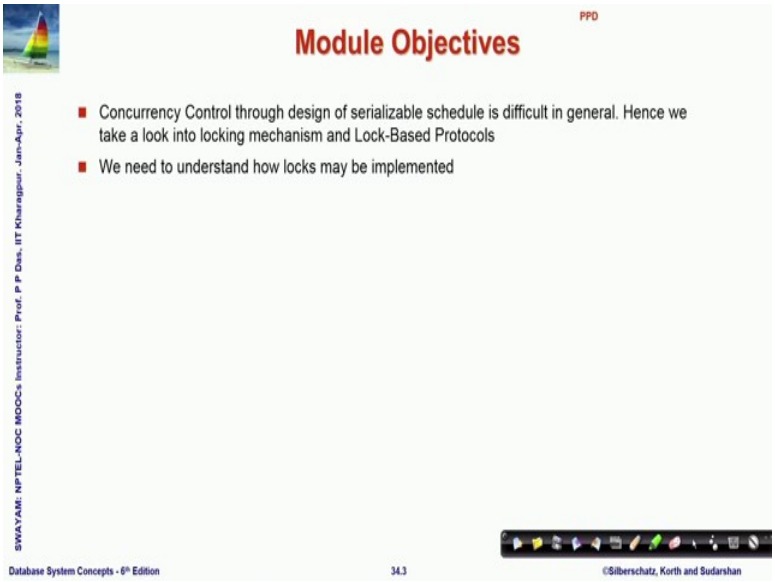
Welcome to module 34 of Database Management Systems, in this module and the next we will talk about Concurrency Control a very key concept of database transactions.

(Refer Slide Time: 00:28)



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.

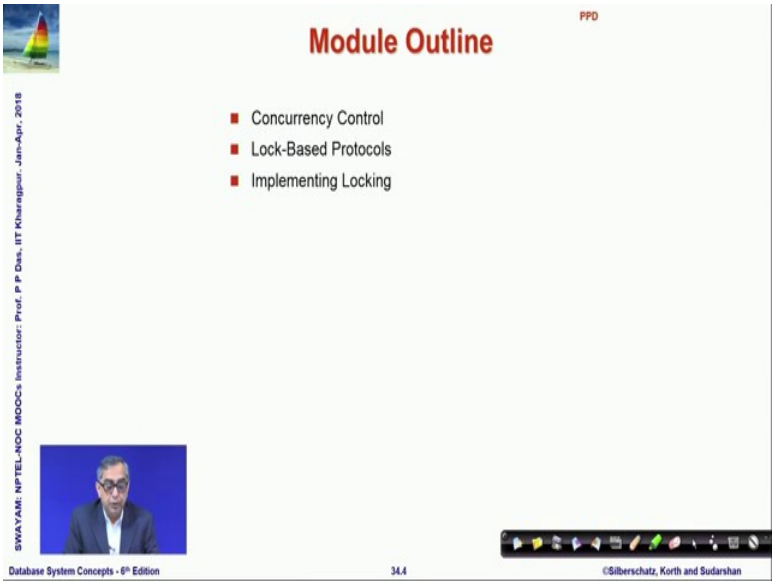
(Refer Slide Time: 00:44)



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 there 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.

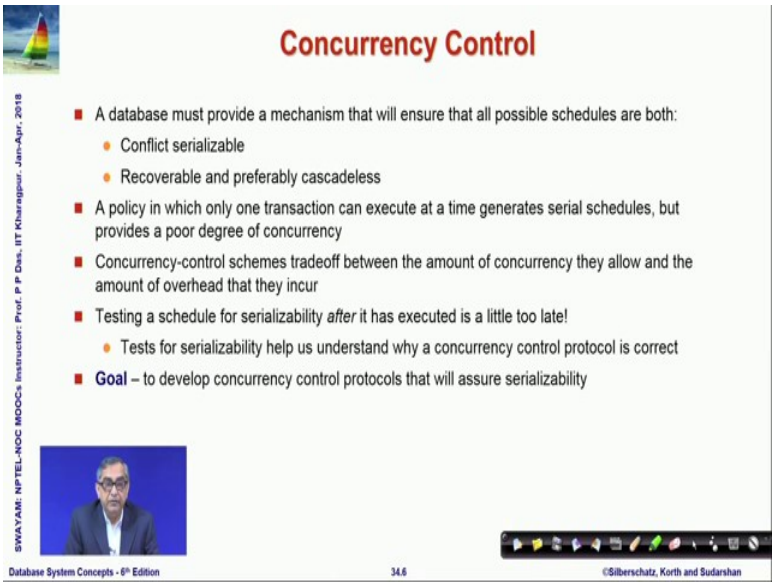
So, how do you prove that or how do you know whether they are conflicts serial, or which of them conflict serializable sets are of view serializable sets and so, on. So, we introduce a different kind of need to have a different kind of mechanism and that is the mechanism of lock that is used.

(Refer Slide Time: 01:40)



So, we will discuss about those aspects issues and the lock based mechanisms.

(Refer Slide Time: 01:45)



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.

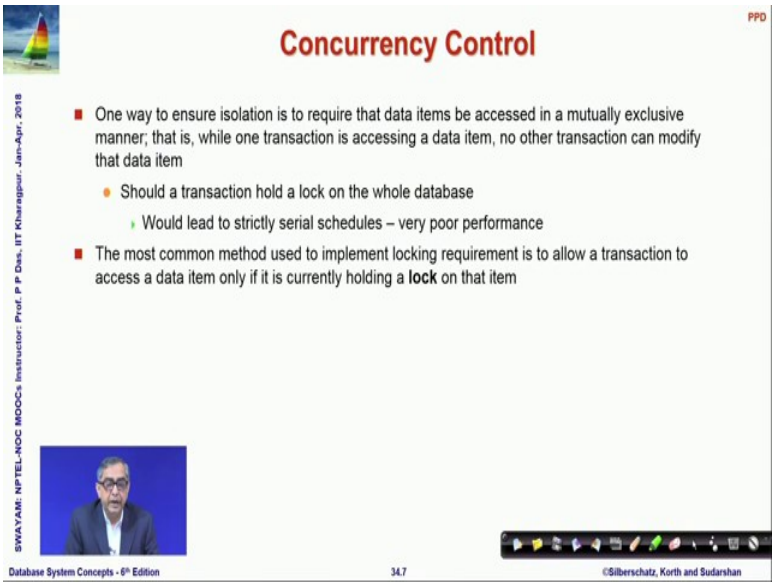
Naturally if we have everything as serial that will happen by default, but that will have very poor degree of concurrency and very low throughput. So, concurrency control

schemes will trade off the amount of concurrency that is allowed and, the amount of over it. So, what will I have to be ensured is I should able to for example, if I say that the 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 concurrence 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)

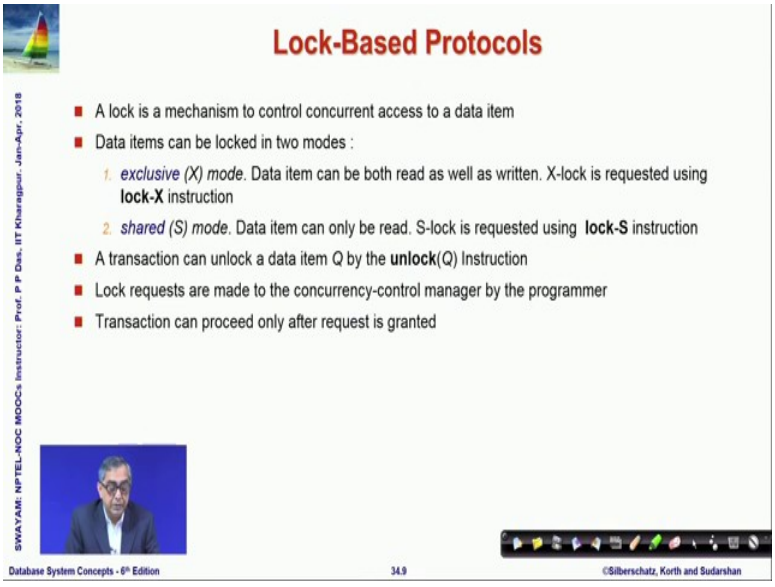


So, naturally what you do you try to see whenever we have conflict, the basic problem of serializability is conflict that is what are you are you reading the right data and, what happens if you inadvertently make changes in a data that has already been read by someone else and so on. So, the why we need 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)



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.

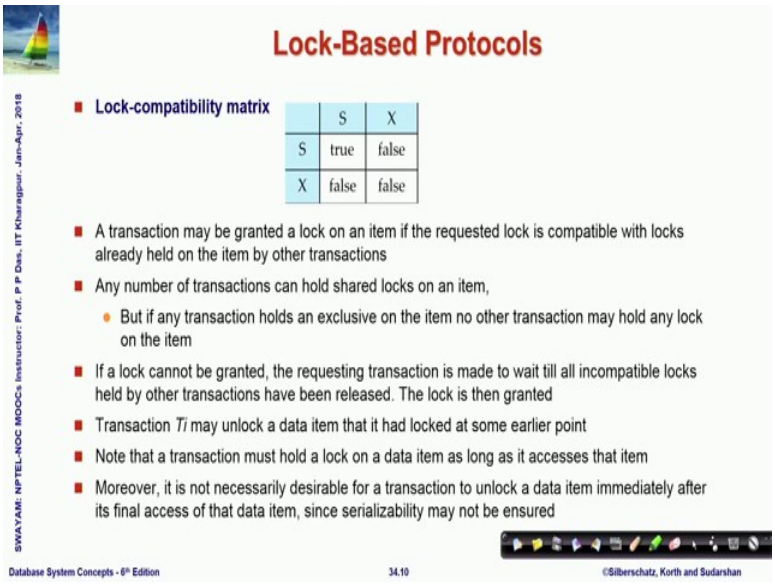
Naturally the exclusive in the exclusive mode, the data item can be read and written both and such a lock is obtained by doing it **lock-X** instruction and in the shared mode the data item can only be read. So, as you can understand why is it exclusive, when you do read write because if two transactions 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)



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

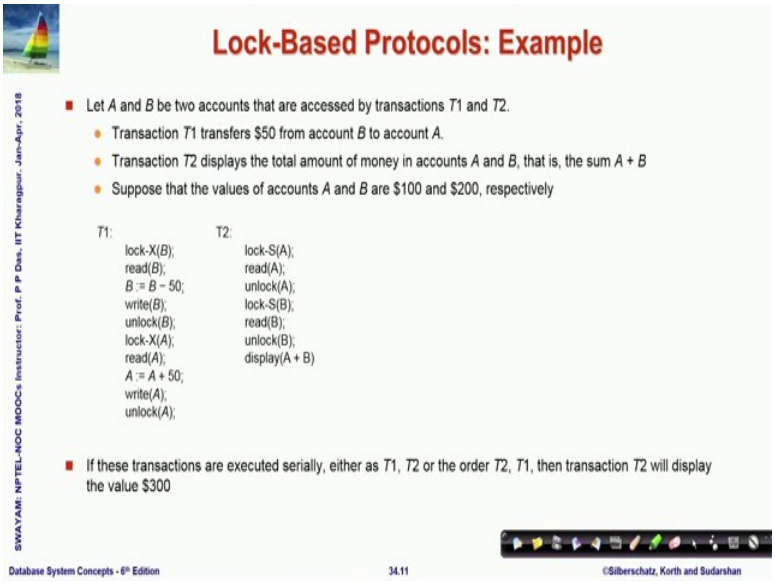
cannot one is reading the value and other is writing, the value is not possible the reverse is also not possible and two transactions writing, the value is not possible those are called said to be the incompatible modes of loss.

So, if the transaction is granted a log, 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 it is 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)



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.

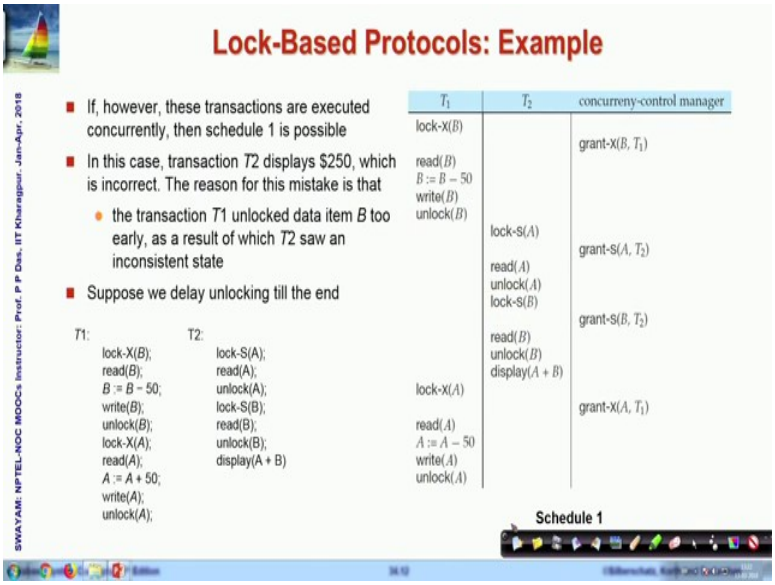
So, it debits B here credits A here. So, it transfers and transaction T_2 displays the total sum of money in the accounts A and B. So, it reads A reads B displays and say initially the transaction initially let us say these accounts have values 100 and 200. So, what this transaction will do it needs 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)



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.

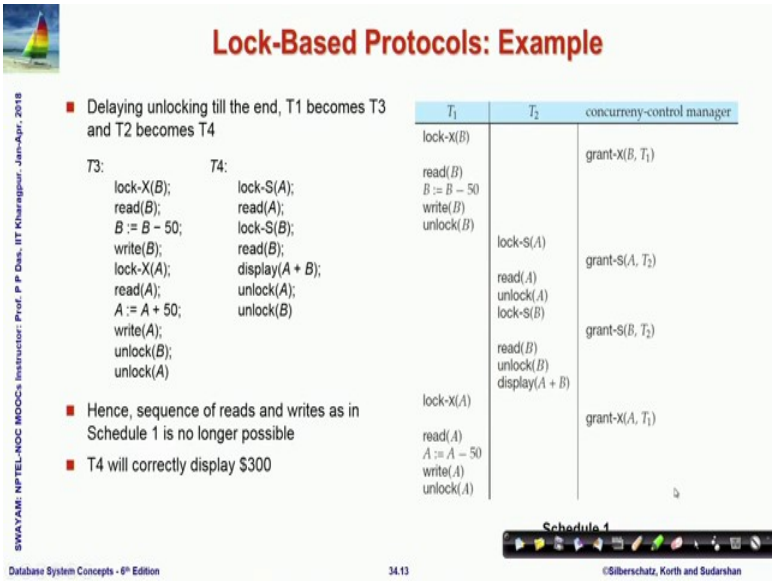
So, as the lock is requested then the request goes to the system that a exclusive lock on B is requested by transaction T_1. So, that subsequently gets granted and only when the grant has happened the corresponding axis can start, but it can be any indeterminate amount of time between the request of 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.

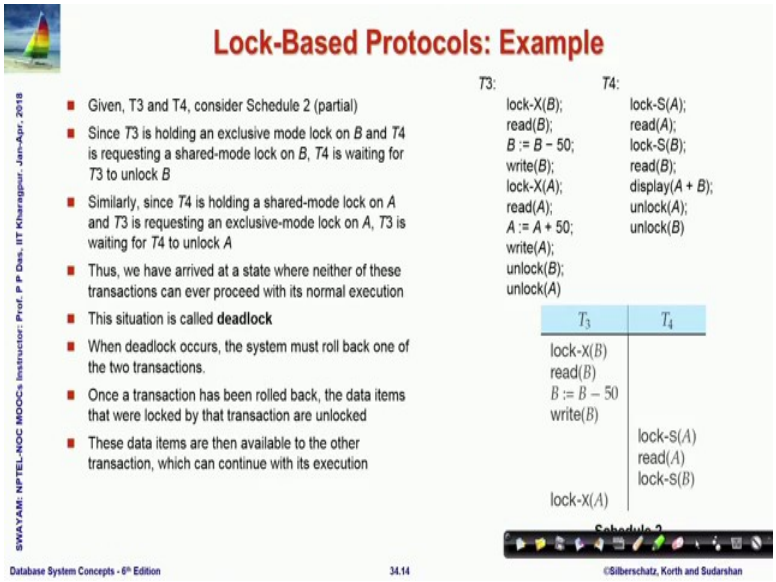
So, it was possible for T_2 to read that value of B which is not what is desirable and we will see that we might want to delay the unlocking till the end, let us see what happens if we do that.

(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)



So, whatever way we actually do the schedule T_4 will always correctly show, that the sum is three hundred dollar. So, here we are again showing T_3 T_4 this is a schedule given schedule 2, which is just given partially. And since T_3 is holding an exclusive lock on B and T_4 is requesting a shared lock. So, if I hold it this is T_3 is holding an exclusive lock and T_4 is requesting for a shared lock.

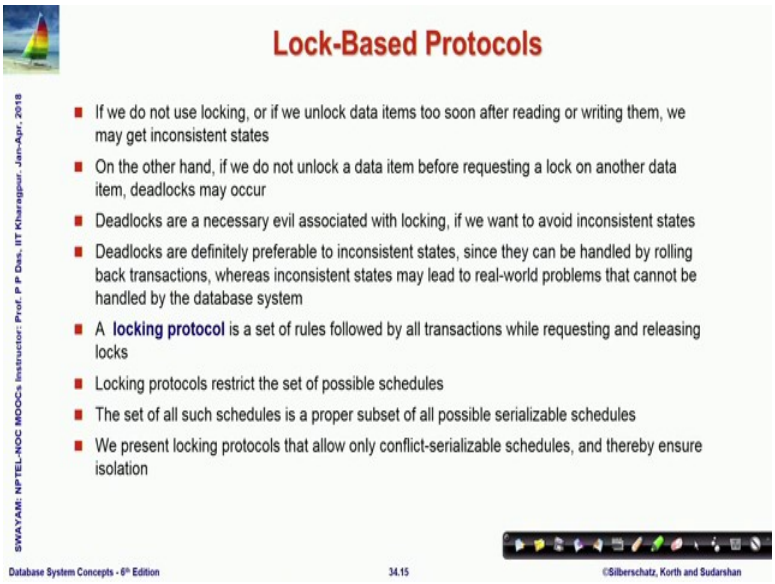
So, T_4 has to wait for T_3 to unlock B before it can actually do that operation. Similarly you will find if you look further T_4 has already got a shared lock to read A. And T_3 needs a shared lock on I am sorry T_3 needs an exclusive lock on A to be able to proceed. So, this one is here. So, T_4 cannot go beyond this point because T_3 has the lock on B and, one is this T_3 cannot go beyond this point because T_4 has that shared lock.

So, what we situation are we getting into. So, we are getting into a situation where, neither of T_3 or T_4 can actually proceed the normal execution, T_3 is waiting for exclusive lock on A and which T_4 has and T_4 is waiting for the shared lock on B, which T_3 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)



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

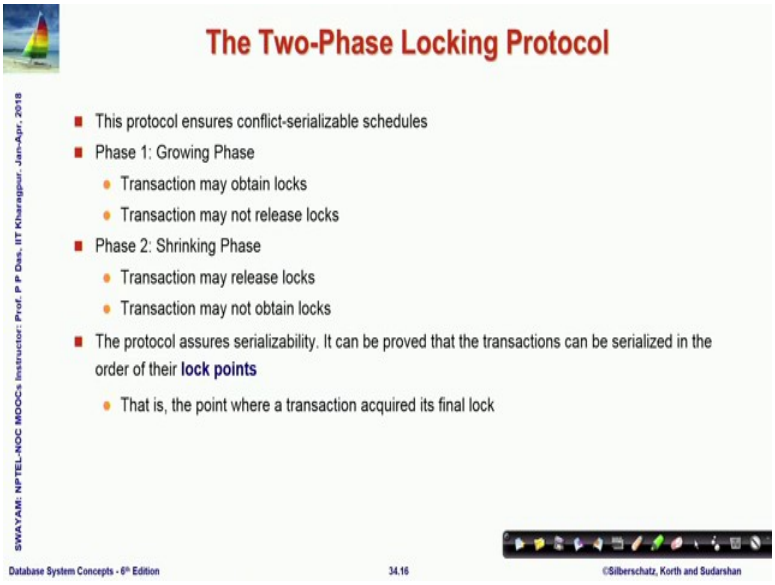
we do not use the lock that we have a problem of inconsistent state, if we do it hold it for too long then there could be problem of deadlock.

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.

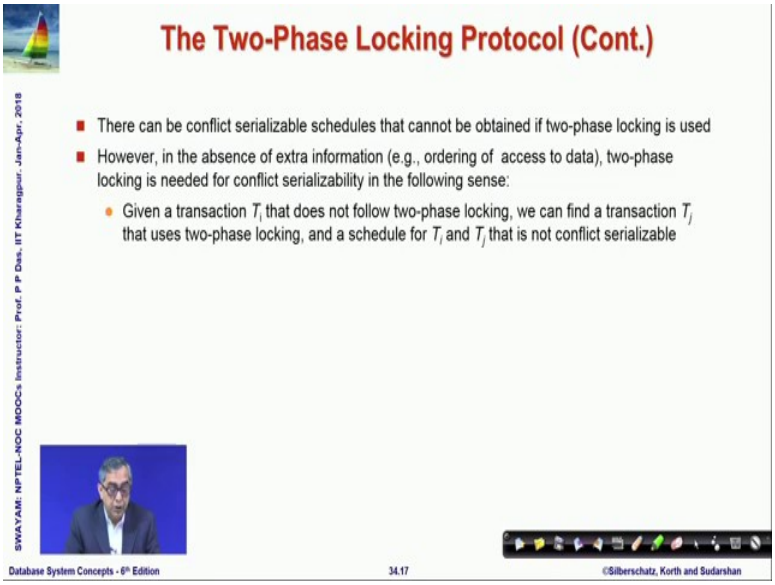
(Refer Slide Time: 21:23)



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.

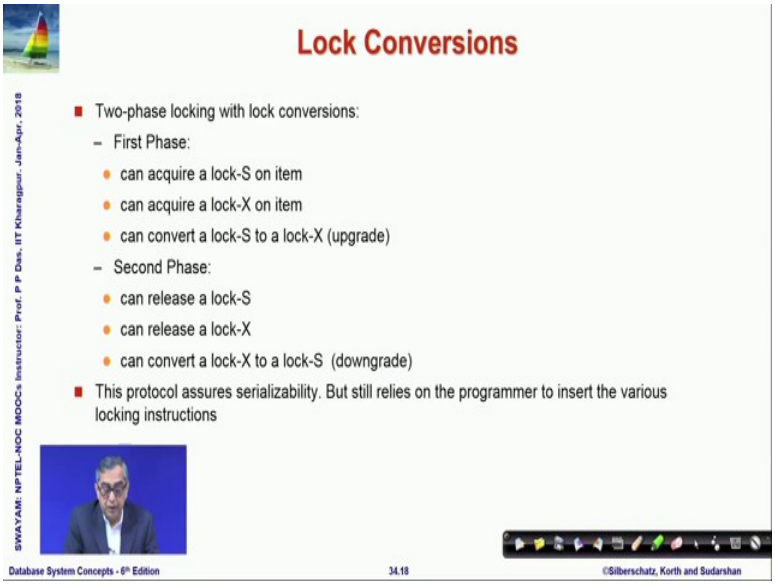
And that is the two phrases phases of the locking protocol. And this ensures so, we are we will not do the proof, but you can look it up in the book or, but you can see through examples that it can be shown that transactions can be serialized in the order of the points where they do the locking. So, these are known as lock points and, that is where the transaction actually acquired it is final lock.

(Refer Slide Time: 22:25)



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)

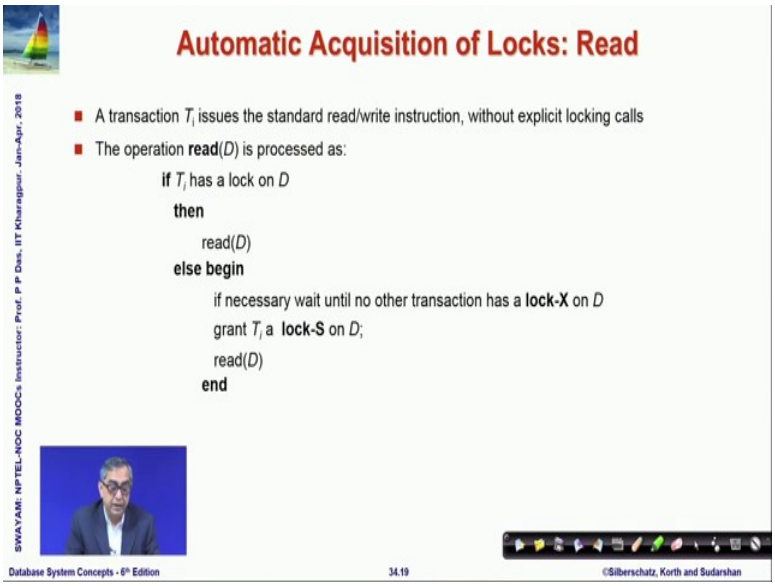


So, you could also in refine the two phase locking with what is known as lock conversion that is you can acquire in the in the growing phase, or the first phase you can acquire a

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.

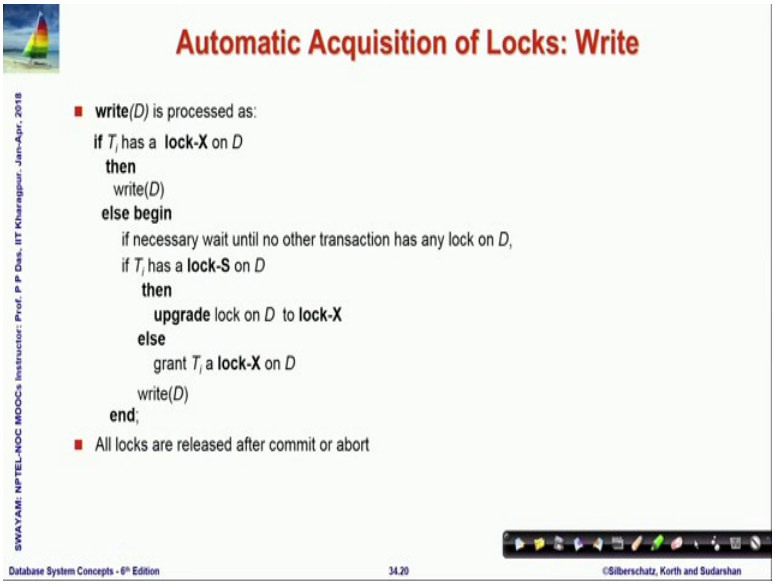
(Refer Slide Time: 23:46)



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.

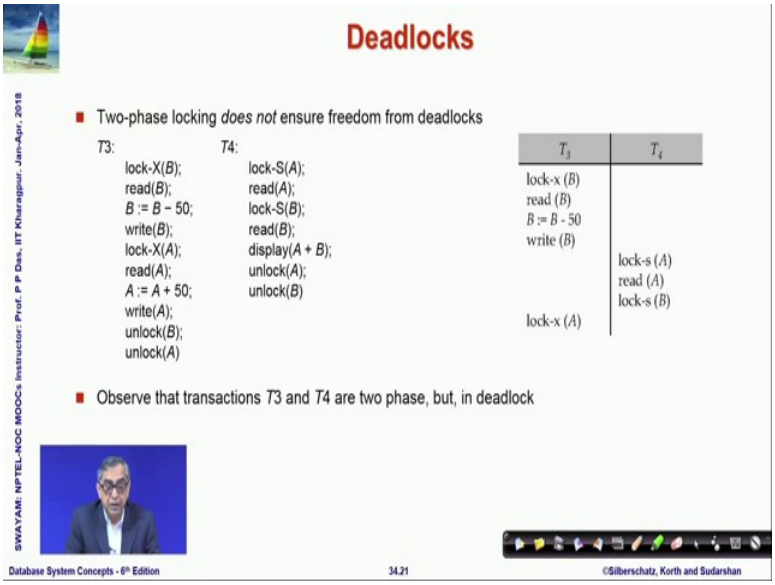
(Refer Slide Time: 24:37)



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.

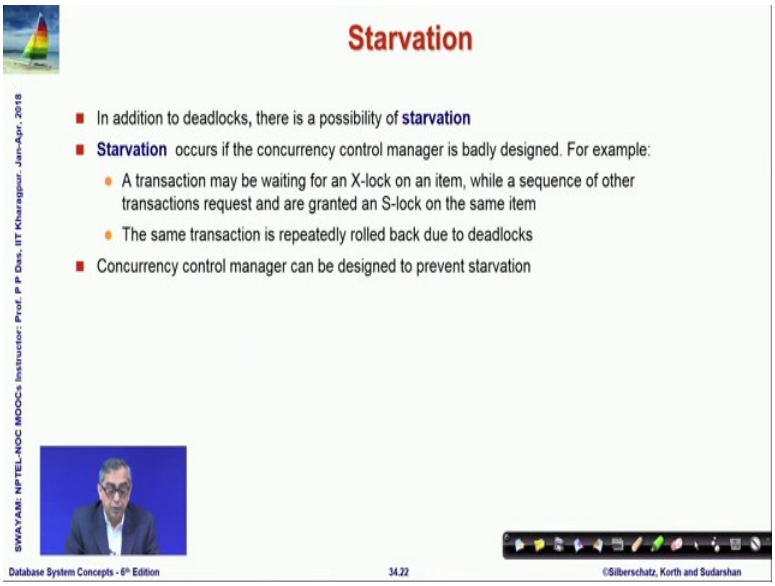
So, as long as some transaction has a lock on D you cannot proceed, but once you come to a state, that you already if no other transaction has a lock, then you see whether you yourself have a shared lock on D, if you have a shared lock then you upgrade it to an exclusive lock, if you do not have a shared 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)



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)



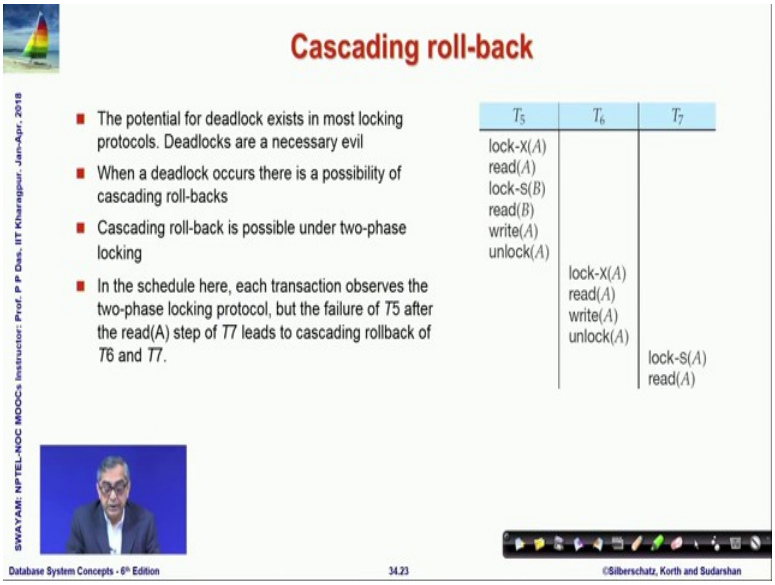
There is another problem that can happen, in addition to deadlock this is a code there is a possibility of what is known as starvation; starvation, occurs usually it occurs when the control concurrency control manager is not a efficient one.

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.

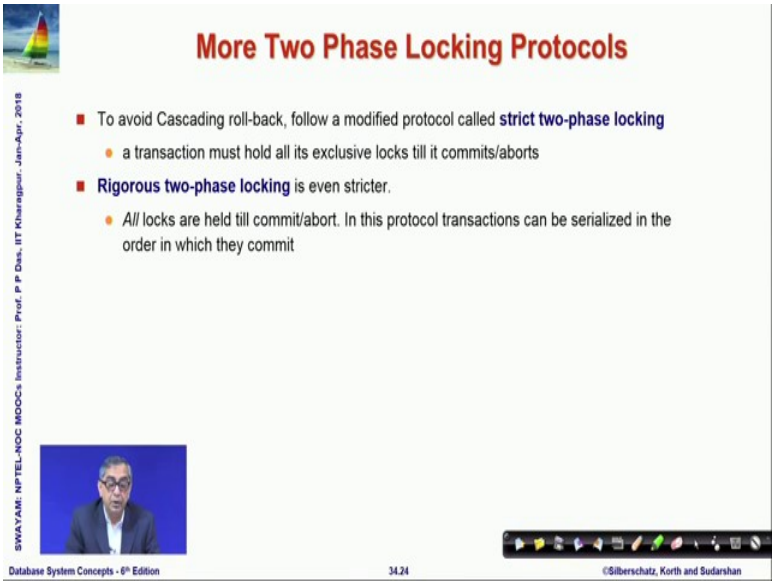
So, if you have a weak strategy in terms of concurrency control, you have you will see that you have had infinite possibilities infinite occurrences, where you could have got that exclusive lock, but you are not being able to get that and therefore, you starve on the data and this is known as a data starvation problem which will also have to be checked while we do the concurrency control policies.

(Refer Slide Time: 27:57)



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 T_5 fails after the read step of T_7 after the read step of T_7, if T_5 fails then it leads to a cascading rollback T_7 T_5 has to be rolled back. So, T_6 will have to be rolled back, so T_7 will have to be rolled back and so on. ah

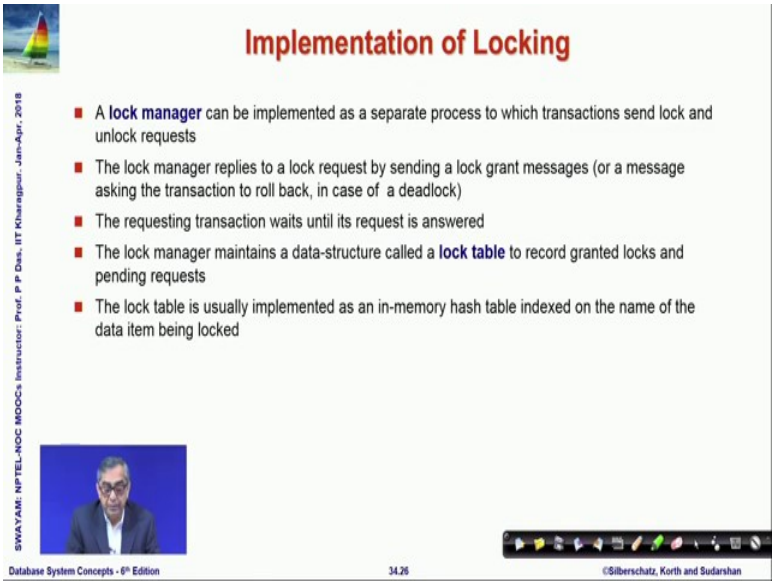
(Refer Slide Time: 28:54)



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.

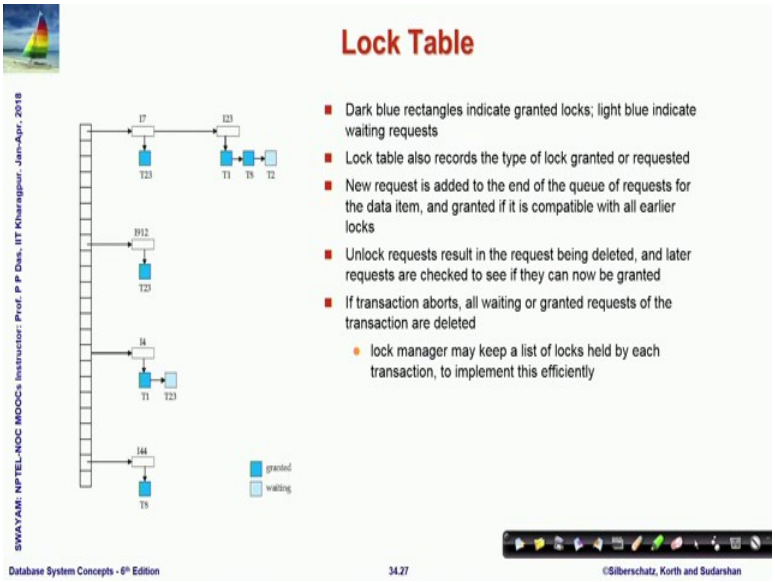
In the strict 1 only exclusive locks are held till commit or abort there is a till the end of that transaction, but in rigorous two phase locking all locks are held till the committed abort, in this protocol transaction can be serialized, in the order in which they do the commit and in that way this 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)



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.

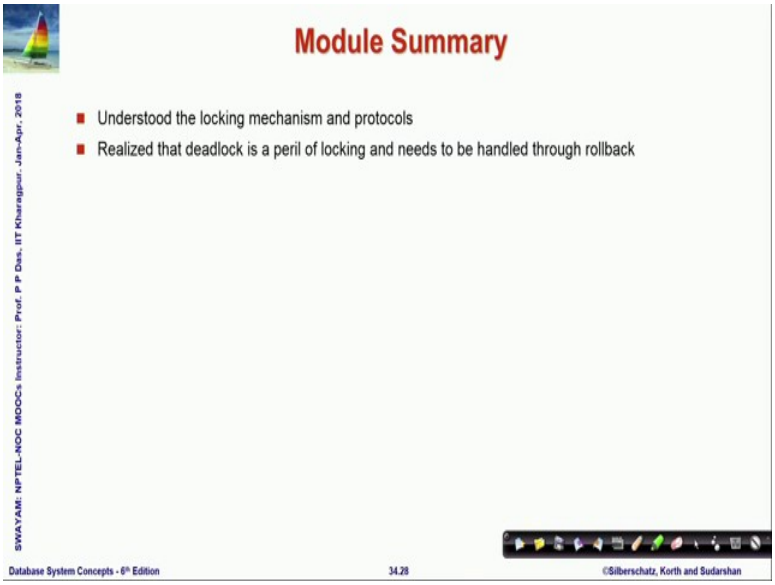
(Refer Slide Time: 31:01)



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_2, 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)



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

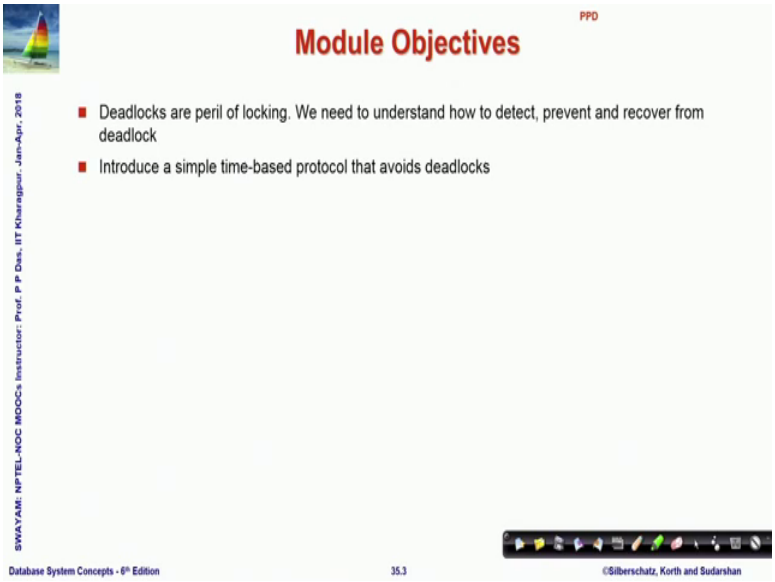
Lecture – 35
Concurrency Control/2

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)

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.

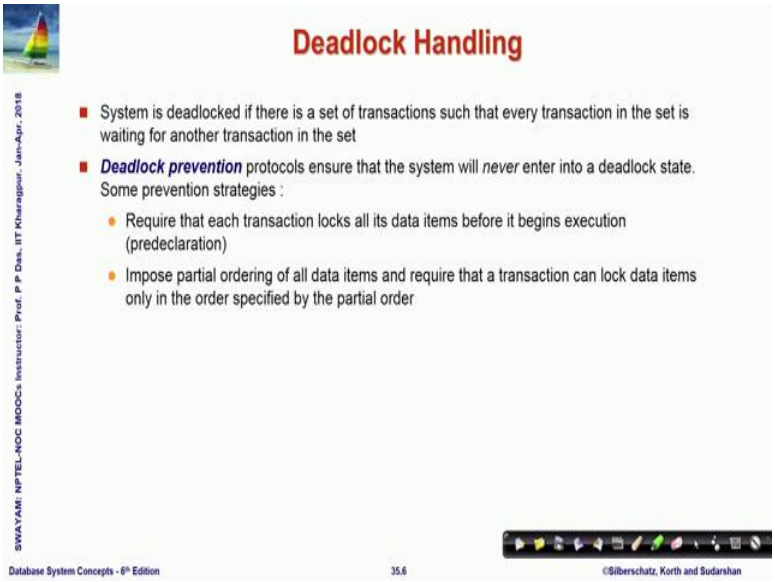
(Refer Slide Time: 00:39)



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)



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.

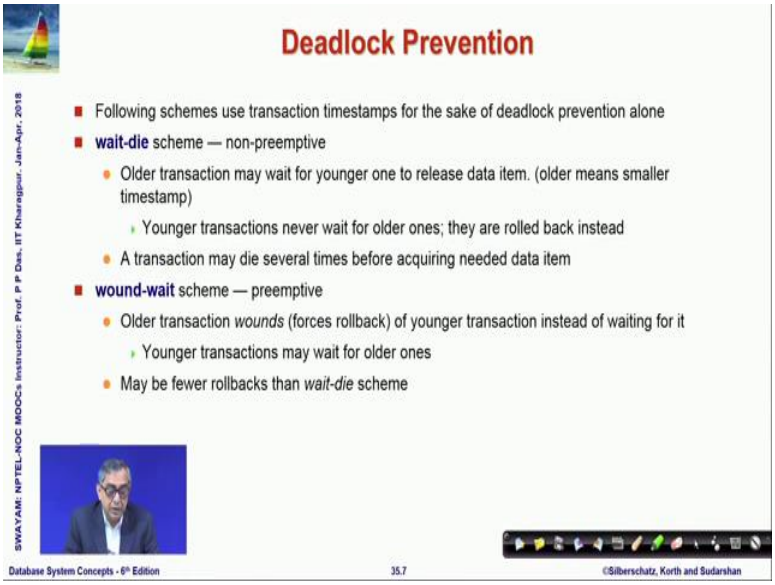
So, can we have strategies so that the requests and releases are done in a way, so that the deadlock will not happen at all. So, I mean fortunately such number of such strategies exist. For example, one strategy which is called a pre-declaration which required that each transaction locks all debt items 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)



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.

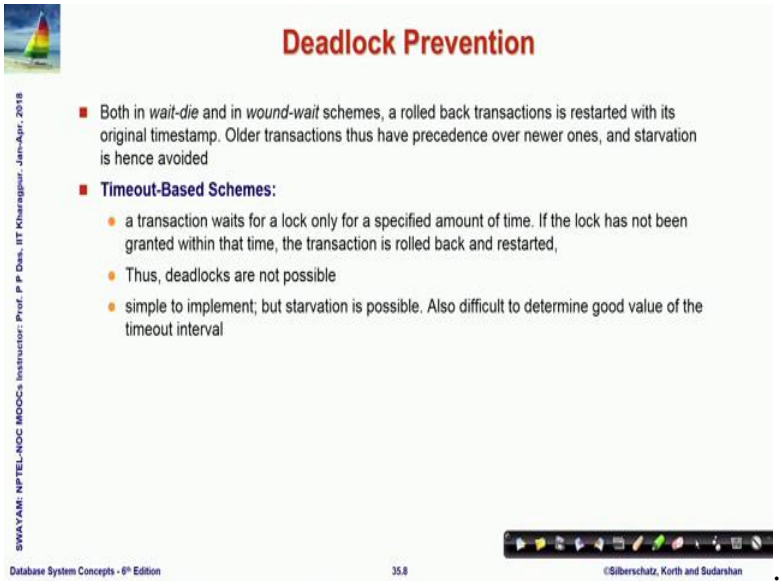
So, what do you do in wait and die, the older transactions because you assume that every transaction has a timestamp. So, smaller the timestamp, older is a transaction. So, older transaction may wait for the younger one to release the item. So, if two transactions are conflicting then the older one will 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)



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.

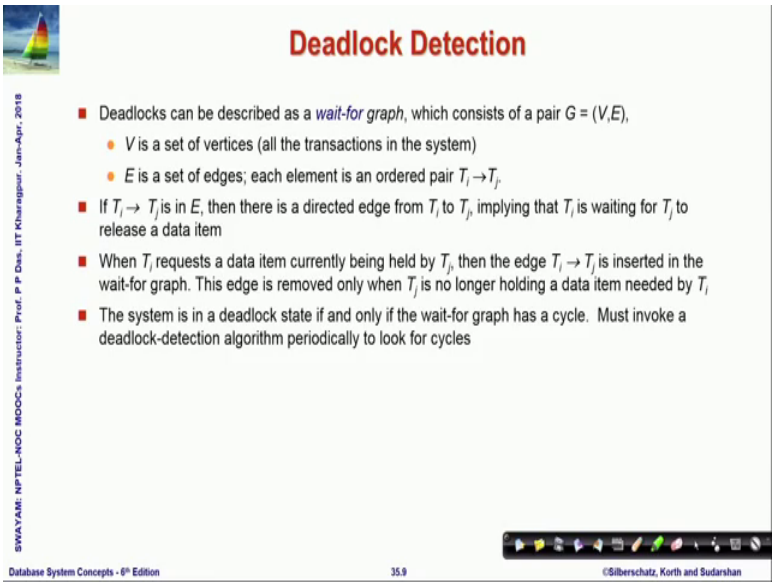
So, now what becomes you are you are actually a new candidate because you have been rolled back in and started again, but you carry your older timestamp. So, your precedence has gone higher because in wait and die, and wound and wait in both actually the older one has a precedence.

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 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 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.

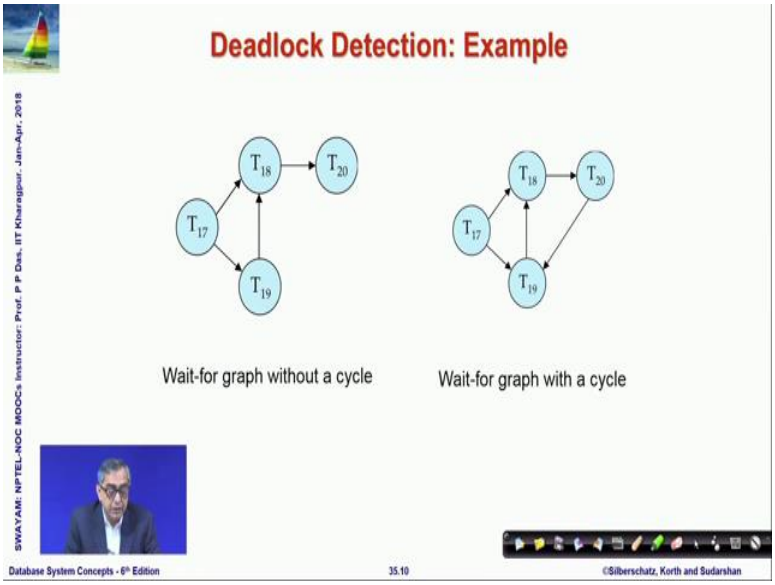
So, this is how this is kind of a dynamic graph the wait for graph is a kind of dynamic graph which will regularly keep getting updated. Now, naturally from the description of this graph, you can understand that a deadlock if a deadlock has to happen then this graph must have a cycle. So, if at any instant 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.

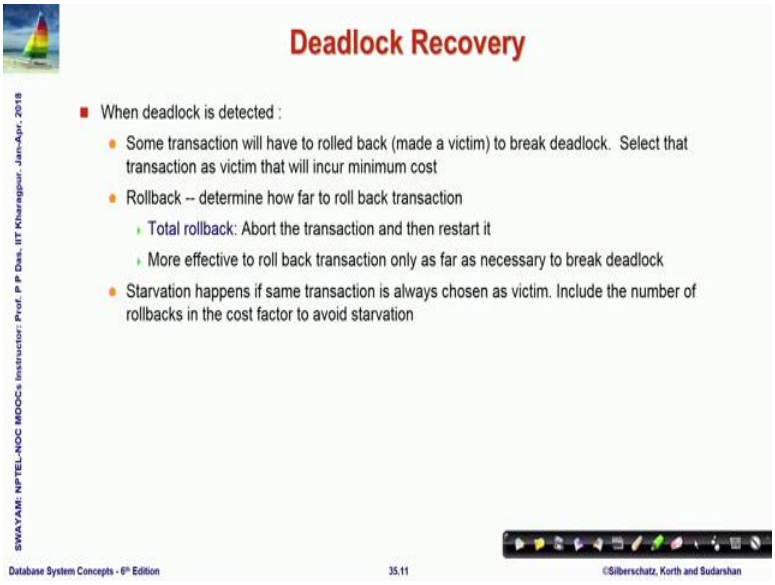
(Refer Slide Time: 11:29)



So, these are examples of the wait for graph. For example, here on the left as you see if you if I if I may point out in the left, if we see that T_{17} is waiting for T_{18} and T_{19}, T_{18} is waiting for T_{20}. So, eventually and T_{20} is waiting for none. So, at some point of time T_{20} will be done and when that is done then T_{18} would be able to proceed. And if T_{18} is able to proceed then T_{19} would be able to proceed, and then T_{17} 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)



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.

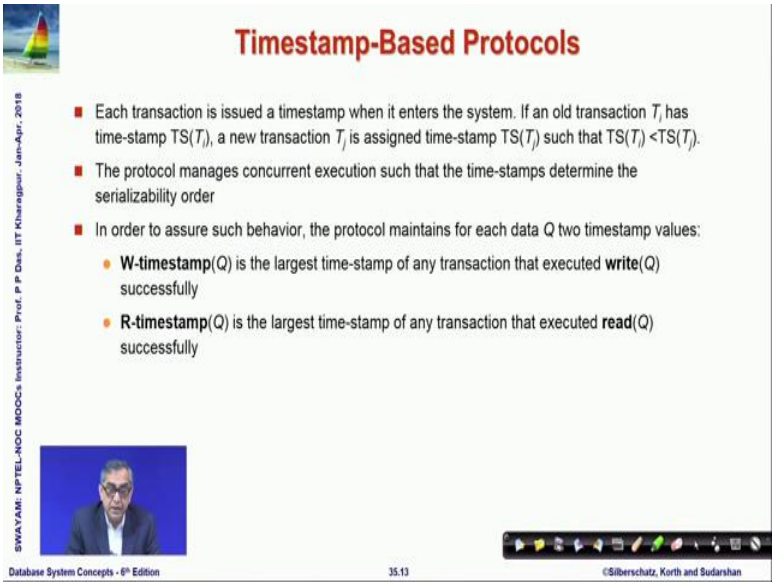
And so it is be more effective to roll back transaction only as far as necessary to break the deadlock, you may not need to roll back everything. Maybe this is this transaction is participating in the deadlock, because it is holding some exclusive lock which it took three instructions before. But before 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)

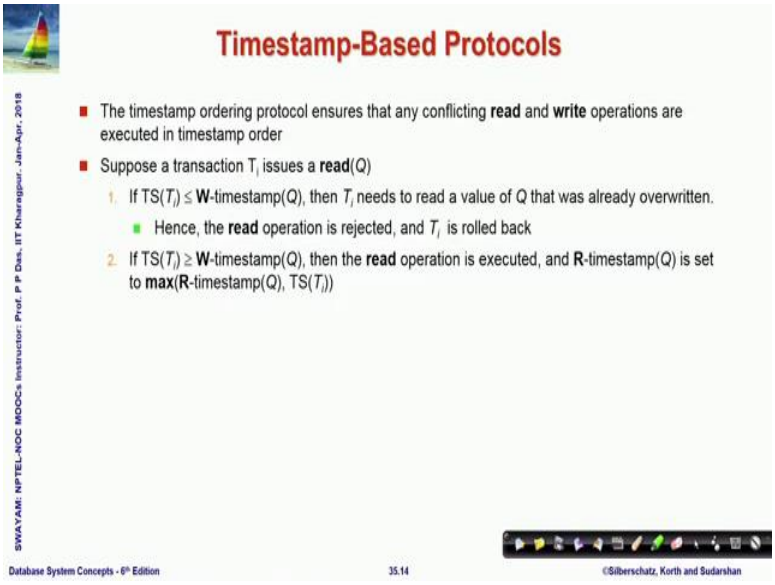


So, having talked about the prevention detection and recovery from deadlocks let us quickly look at a simple time-based protocol in contrast to the two phase locking protocol we had earlier. This protocol does not lead to deadlock. So, what you do in here is each transaction is issued a timestamp when 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)



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 <= W-timestamp (Q)** which means that **W-timestamp(Q)** is the latest write. And TS(Ti) 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.

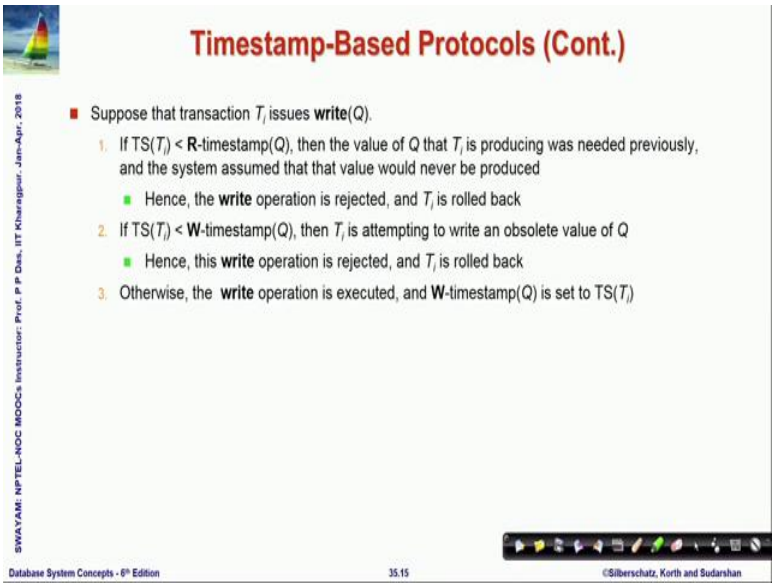
If it is in contrast, if the timestamp of the transaction is greater than the latest right time W-timestamp(Q) then the read operation is executed and since we are doing a read operation. So, this becomes the latest read operation and therefore, the read timestamp **R-timestamp(Q)** is set to the maximum 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)

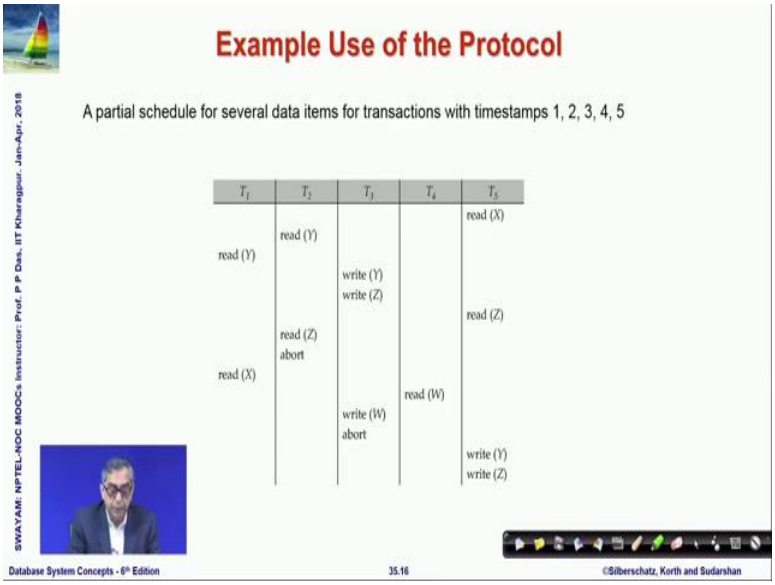


Write is a little bit more complex, but we can reason in the same way. So, if T_i issues a write (Q). And if the timestamp of **T_i < R-timestamp(Q)** that is if this transaction is it is less so it is older than the read timestamp that is it is older than the transaction that read Q last, the youngest 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.

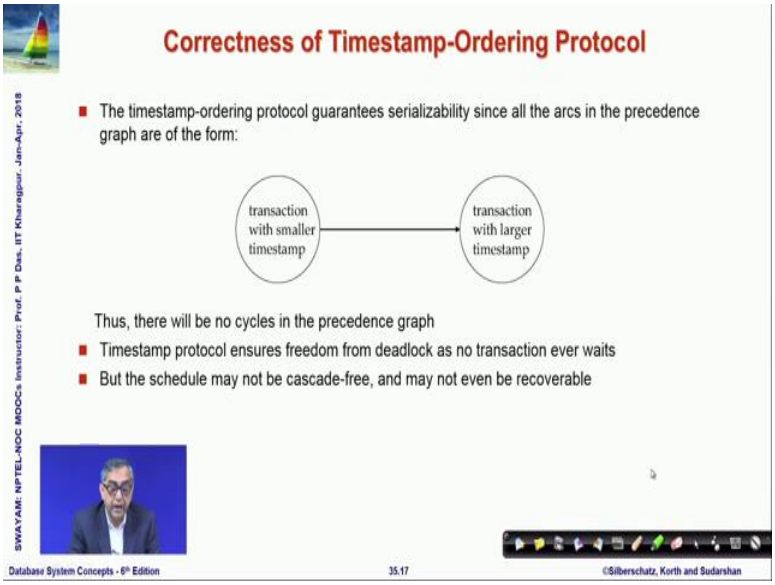
(Refer Slide Time: 20:36)



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.

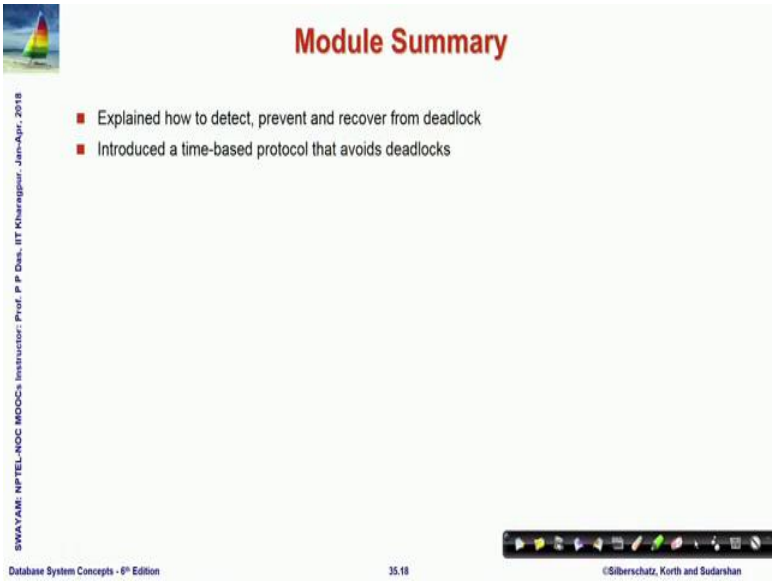
(Refer Slide Time: 22:08)



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)



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.

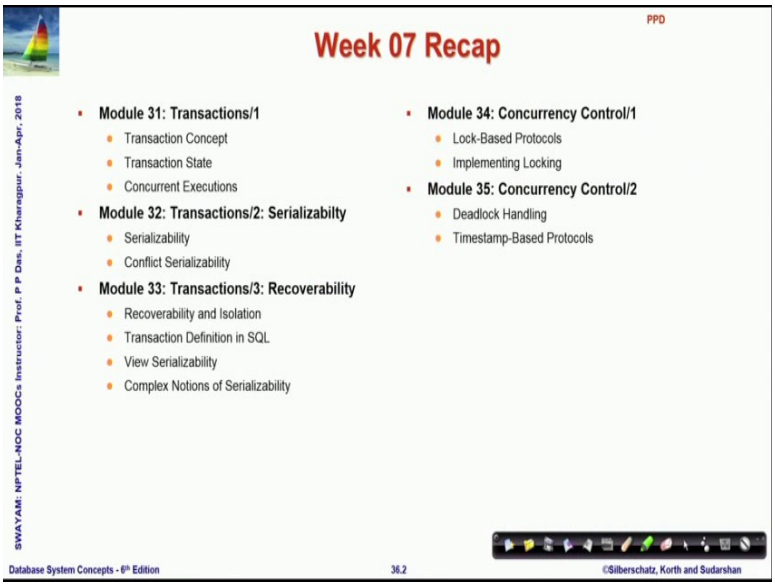
And on that study we have presented a simple time-based protocol which maintains the timestamp of transactions to decide the ordering in terms of read and write and deciding as to whether you should continue doing a read or write or you should abort the read and write attempt; And thereby ensuring that 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

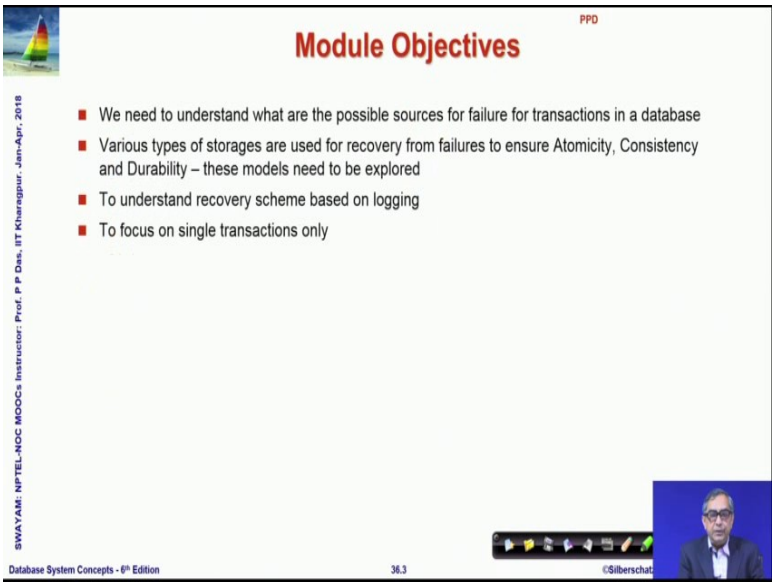
Welcome, to module – 36 of Database Management Systems. In this module and the next, we will talk about recovery in databases.

(Refer Slide Time: 00:27)



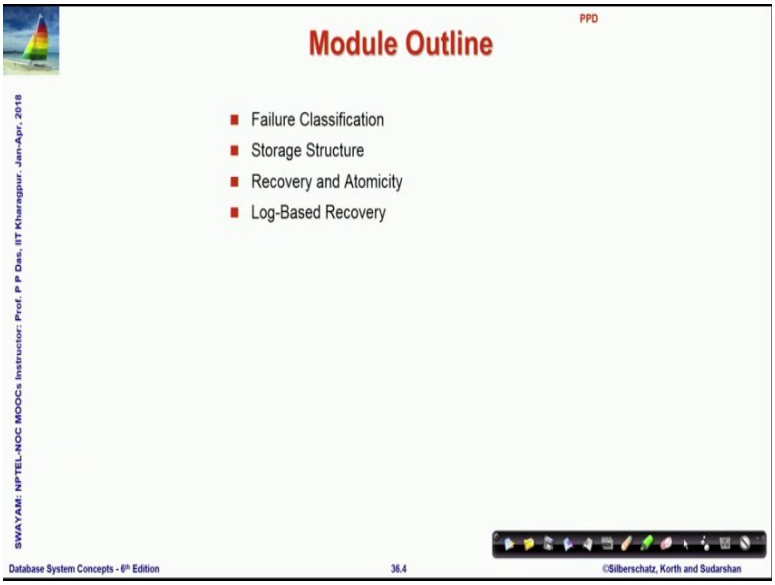
In the last week, we have talked at length in terms of the transactions and the concurrency control. And we will see how the acid properties of the transaction can be fulfilled using the different recovery schemes.

(Refer Slide Time: 00:41)



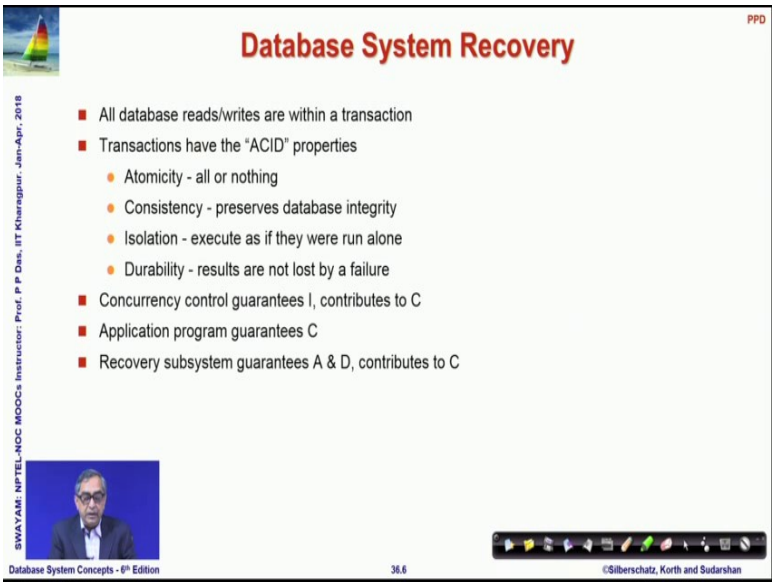
To, specifically we will try to understand the different sources of failure and how the recovery can be facilitated by different storage structures particularly those different models of volatile and nonvolatile storages and we will take a look into recovery schemes that are based on logging mechanism and for this module we will focus only on single transactions.

(Refer Slide Time: 01:11)



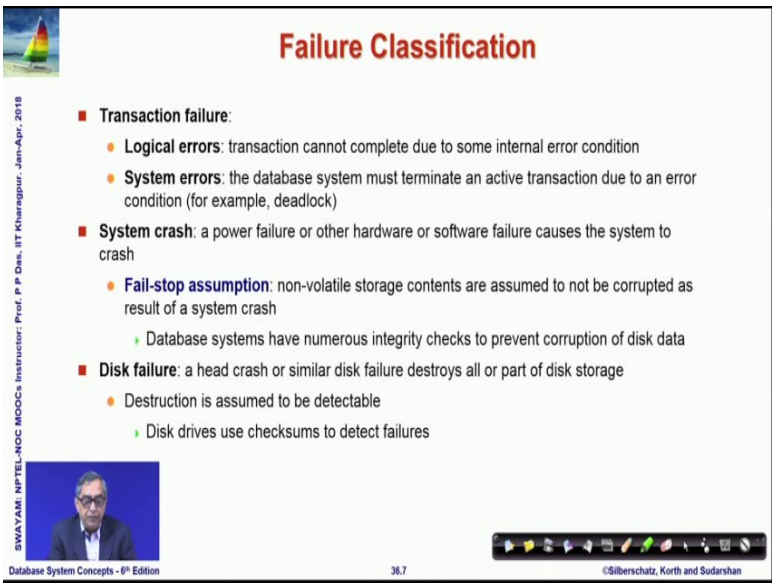
So, these are the topics that will cover.

(Refer Slide Time: 01:17)



So, we 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 guaranty 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)

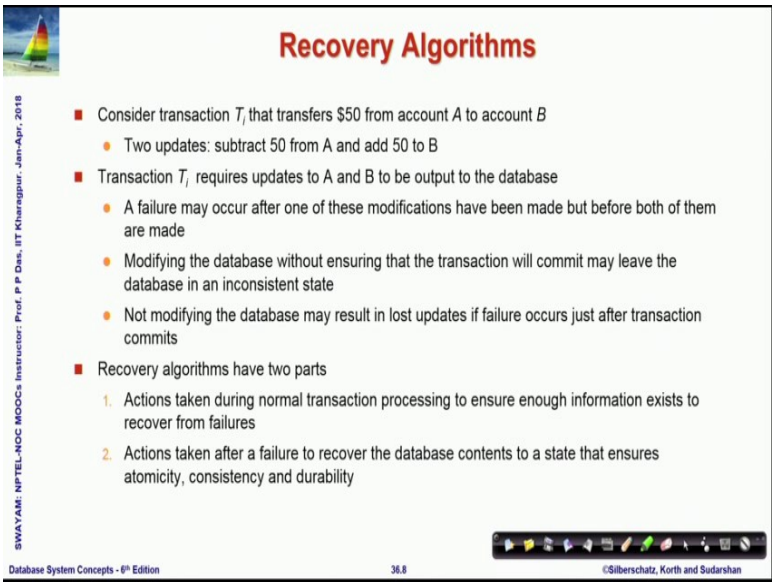


So, let us look at if we are talking about recovery and the phase of failure. So, let us look at what are the generic types of failures that can happen one is the type transactions can fail. A transaction can fail due to logical error due to some internal error or it might fail due to some system error. 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)



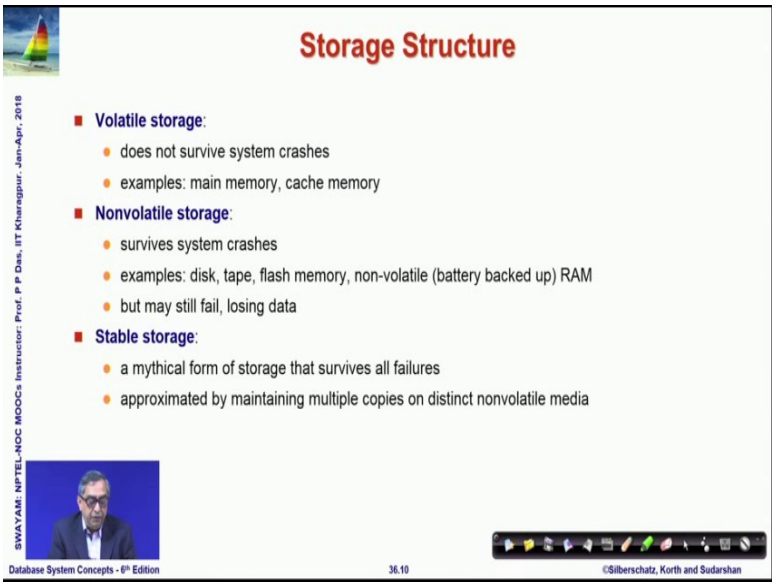
So, in view of that ifs one or more of this failures happen then we need mechanisms to recover from that let us consider a very simple situation of a transaction which we saw earlier to that a transaction T_i transfers dollar 50 from account A to account B and

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.

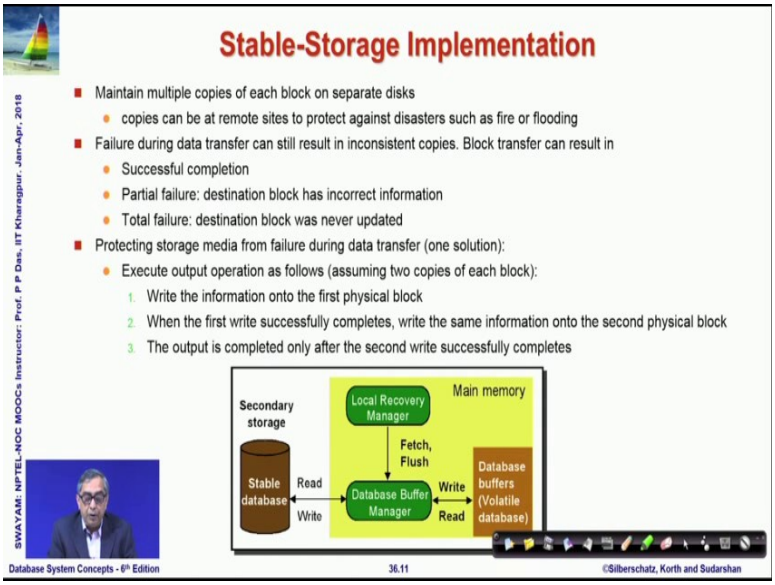
(Refer Slide Time: 05:30)



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 stables storage would be assumed to be one available component in the database system for making the recovery systems work.

(Refer Slide Time: 06:21)



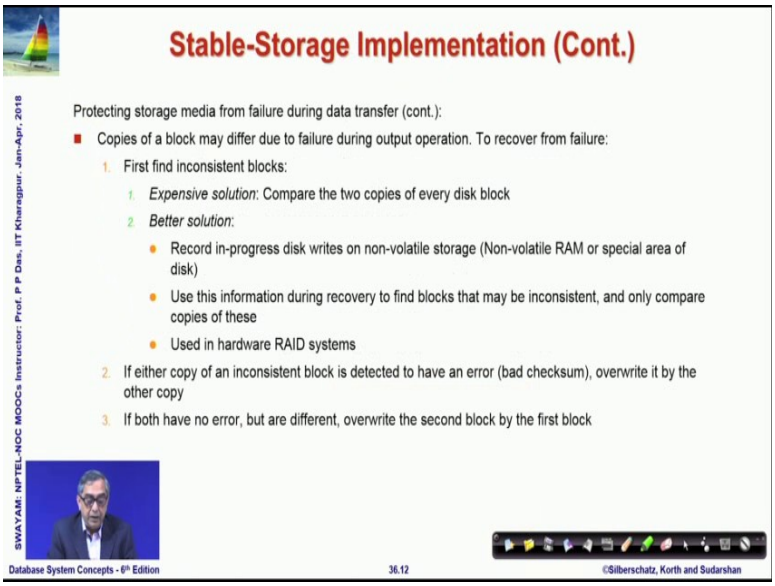
So, as you can see in this diagram below. So, we are trying to explain more of what is the stable storage. So, this is a on the secondary storage so, you have a stable database. So, kind of approximates that it will never fail whereas, on a on a routine basis things happen in terms of database buffers 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 ba 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)



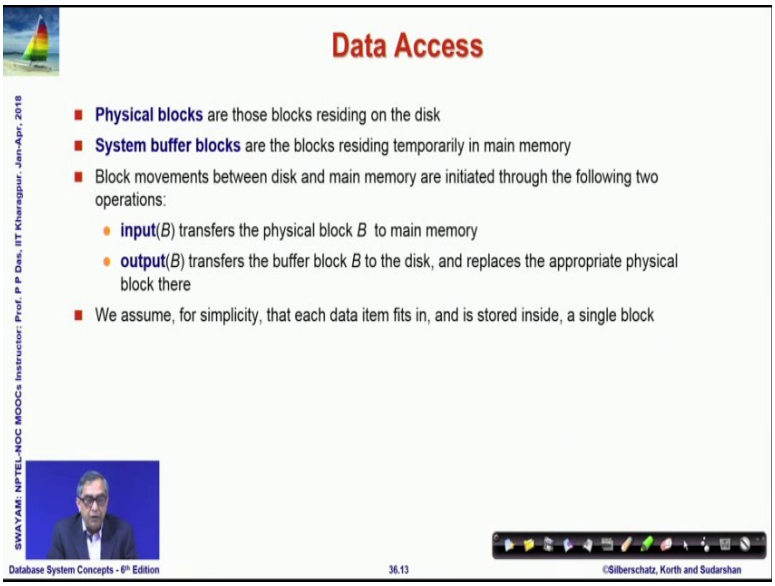
Now, to protect against that ba that happens if during this transfer with during this write if some output operation is some failure happens. So, to recover from that you need to find out what are the blocks which are inconsistent, because we have kept two or more

copies so ba 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 inconsistence. Now, this is 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 the 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)

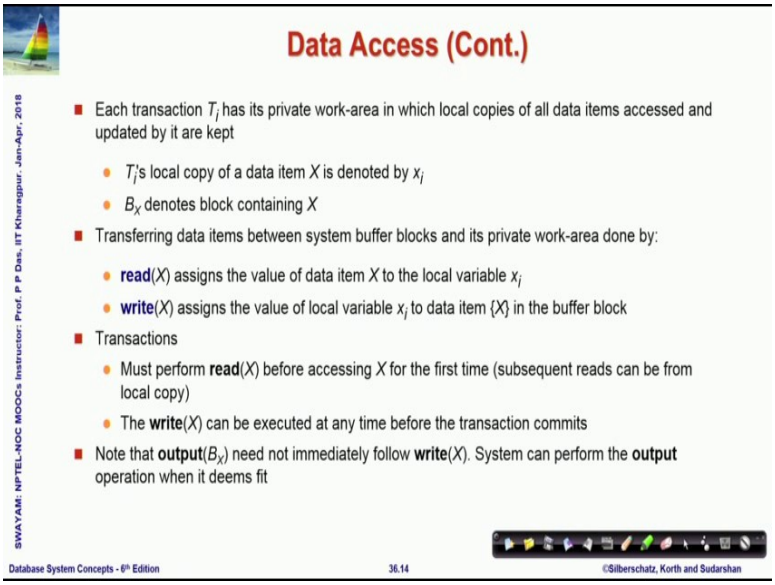


Now, to make this kind of a mechanism work we you resort we have very simple module of data access. We assume that there are physical blocks on the disk that that are on the non-volatile permanent storage and that is where finally, you want your data to decide,

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 you data over multiple blocks.

(Refer Slide Time: 11:40)



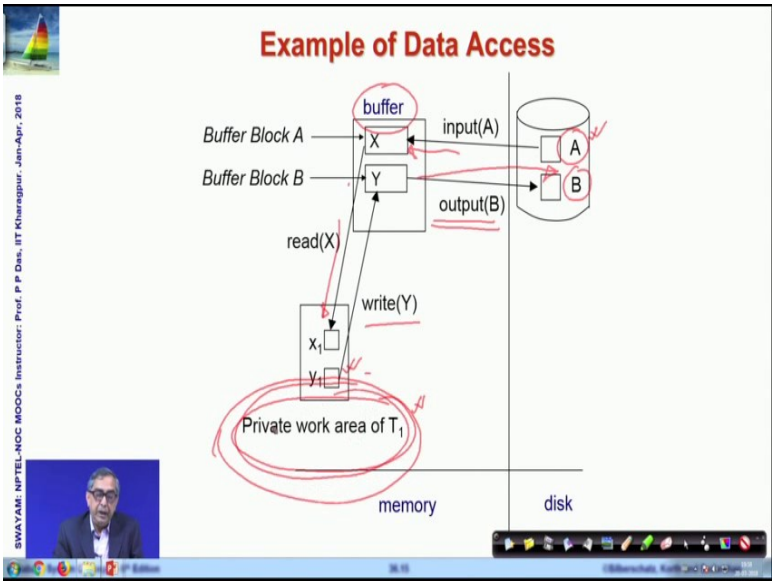
Now, the other part is each transaction on the other side is assumed to have a private work area. So, in the private work area that transaction actually gives local copies and these local copies say you have a data item X, so, you say that for transaction T_i the copy of that data item X is x_i and say, 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 T_1.

So, to process of read I mean if T_1 wants to if T_1 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 x_1 in its private area, it will this is where it will do the work. This is a private area where T_1 will do the work and possibly it has generated a write item y with which needs to go back to the

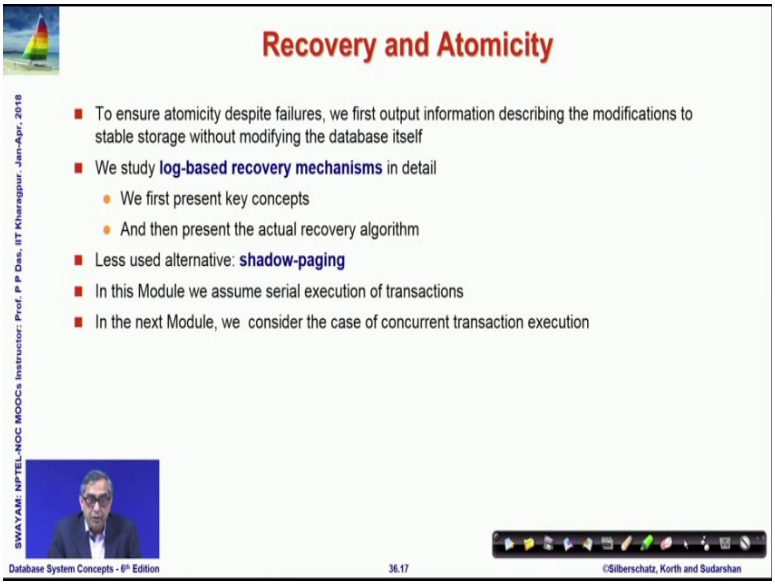
disk. So, again we will do a write to the buffer area and then at a later point there will be an output which will take this Y back to the disk.

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 the 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)

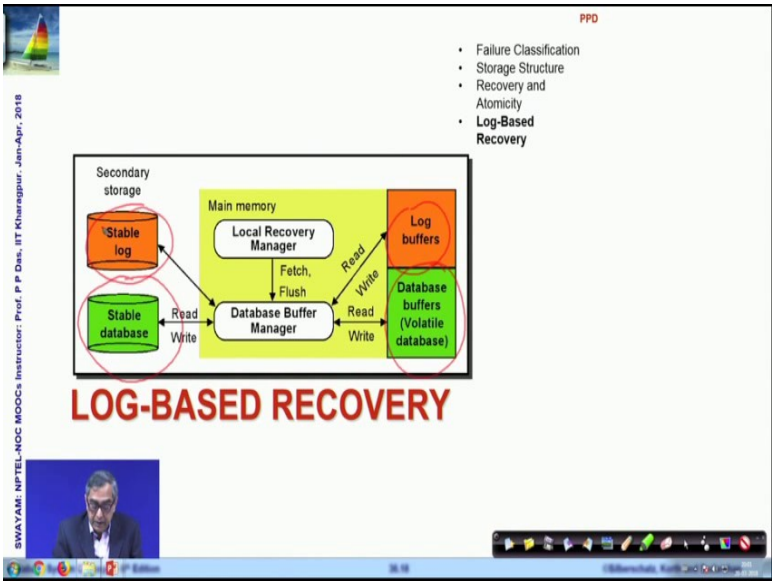


So, to ensure a atomicity in the phase of failure we need to output information describing the modifications to stable storage with input modifying database itself. So, what we are saying that to be able to recover that we should write the changes to the stable storage you recall that stable storage 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.

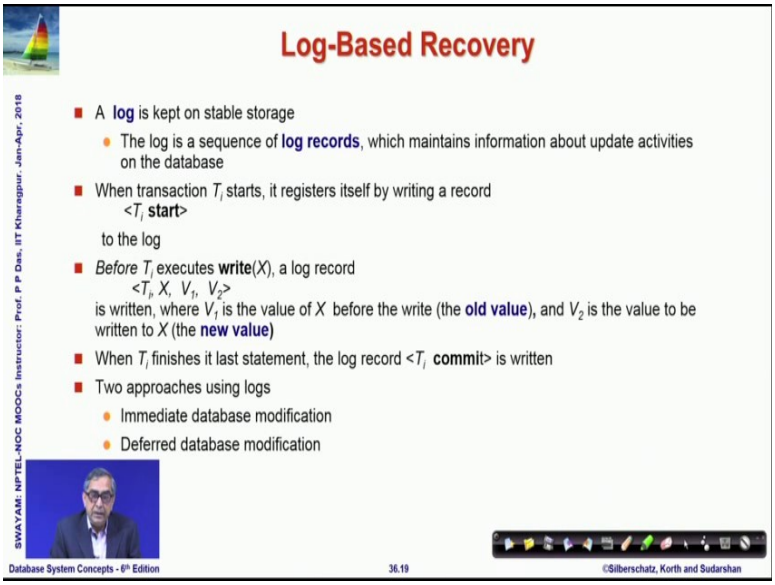
(Refer Slide Time: 17:29)



So, now let us talk about the log based recovery mechanism. So, ma in the log based recovery mechanism you can 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)



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 a 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.

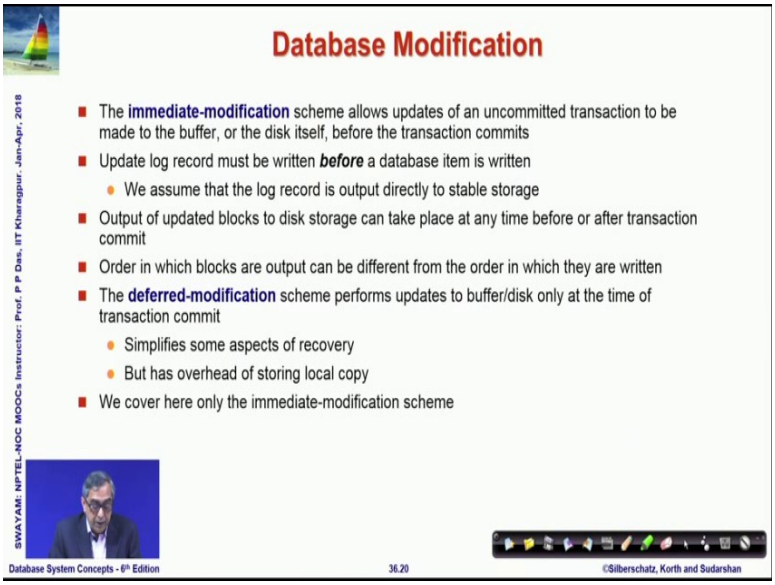
So, here if we look carefully this is the idea of the transaction X is the data item that you want in to write V_1 is the current value of the data item which kind of we can say is the old value and V_2 is the value that we want to actually write. So, here you can see that we are clearly keeping a track 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)



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 **<T_i, commit>** 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.

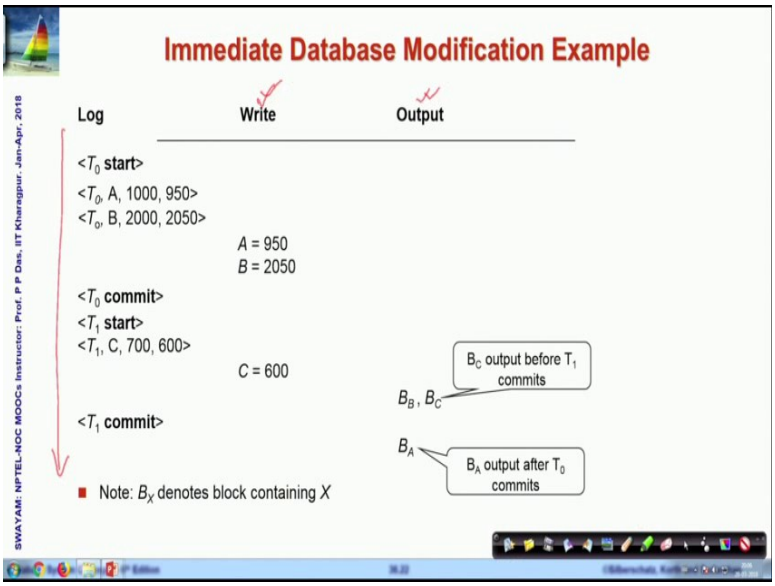
In the deferred modification scheme the change updates are performed to buffer and disk only at the time of transaction commit not any time before that. So, that simplify some aspects of recovery, but it has other issues. So, we will not talk about this scheme, just know that there is an alternate scheme for doing things.

(Refer Slide Time: 22:03)

So, now formally speaking what is transaction commit? A transaction commit is said transaction is said to have committed if it commit log record is output to the stable storage. That is T_i commit has gone to the stables 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 that 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 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 it 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.

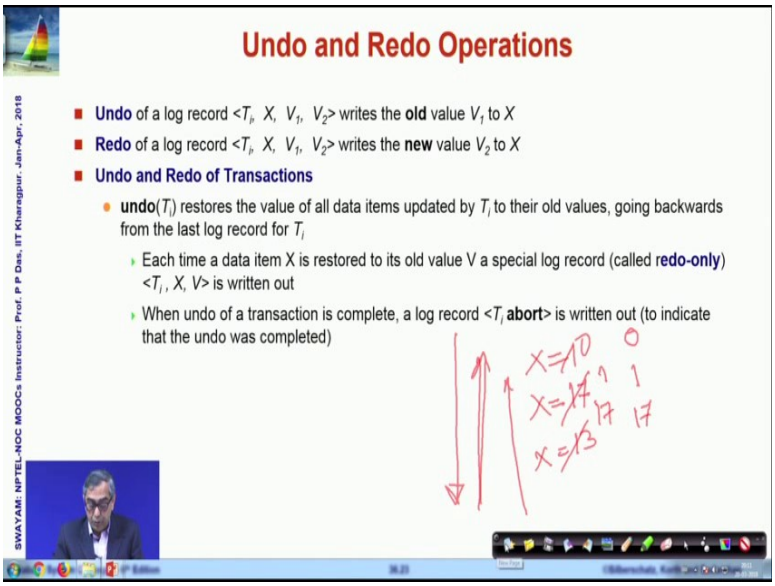
In the meanwhile and at this stage, in the meanwhile these blocks have been output. So, they have actually been written the disk and you can understand that this block B B is a

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)



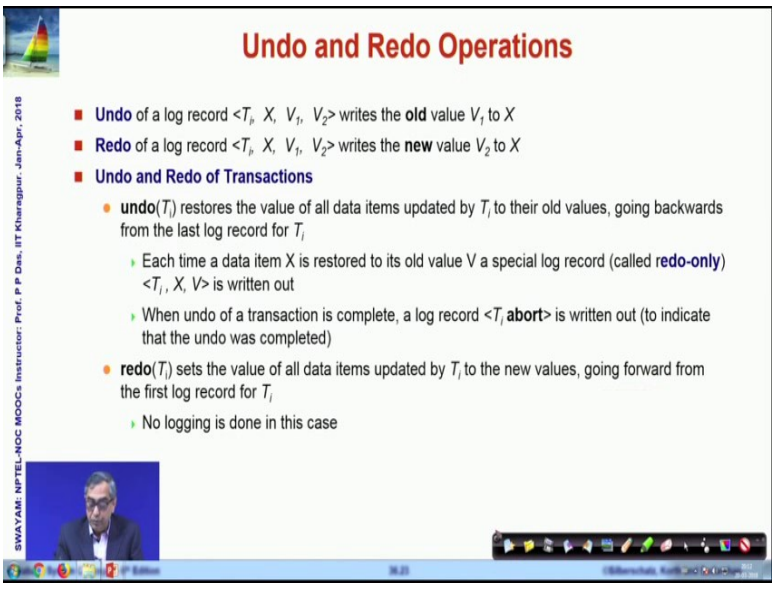
Now, the question is we have written the logs. So, what is the use of those logs? Naturally, the use of those logs are in terms of two operations to which we say are undo operation and redo operation and undo operation is one which basically undoes the operation the effect of an update. So, while ma 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 will have to do that in the reverse order, because it is quite possible that X got 1 here then at a some at a some later 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 transactions 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.

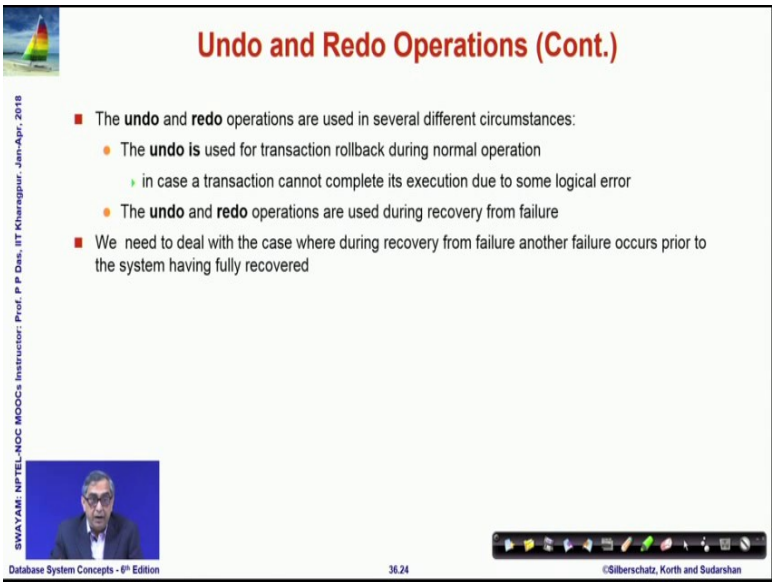
And, every time you restore you write that you write that out as a record which is known as redo, redo only record. So, you can see that here you are not trying to remember the original value you are just writing the value that you have written out in terms of the undo operation that is the old value 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 **<T_i , abort>** is written out which says that the undo is actually over. So, this is the undo operation.

(Refer Slide Time: 28:36)



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.

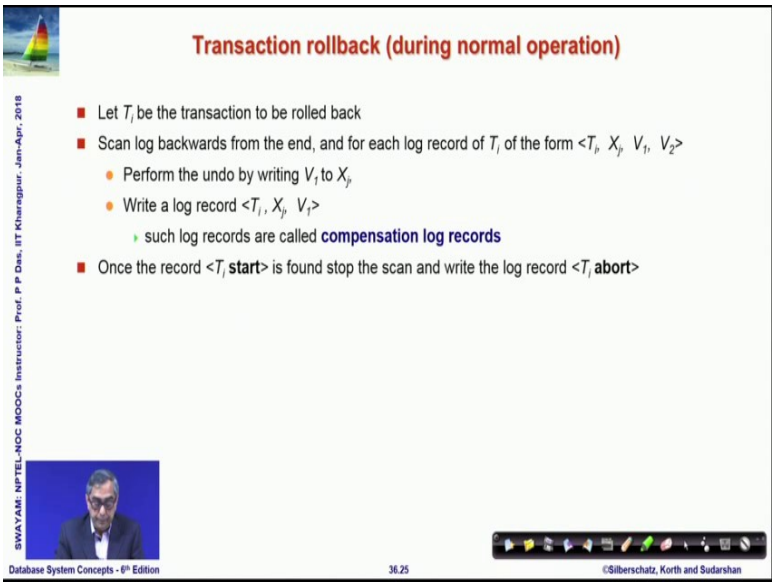
(Refer Slide Time: 29:04)



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?

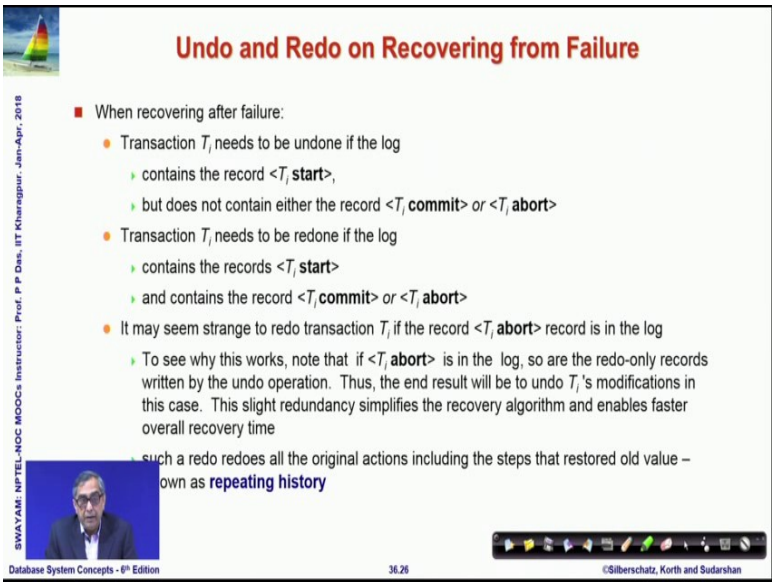
(Refer Slide Time: 30:04)



So, first let us discuss what happens when you roll back at transactions 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 in 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 **<Ti , start>** log record when you face come across that you write log record **<T_1, abort>** 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)



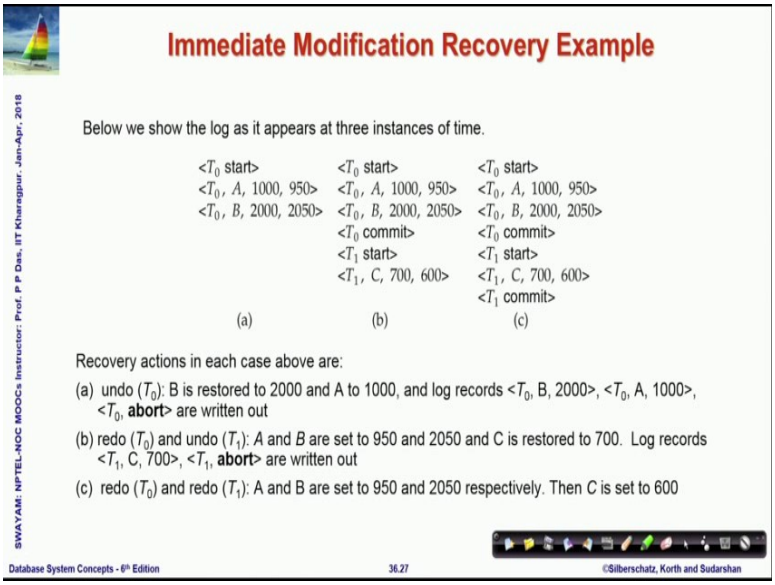
In the other case if you are recovering from a failure if there has been a failure then you do something which needs to be understood carefully. So, the transaction T_i needs to be undone if the log contains the record start **<Ti, start>**, but it does not contain either **<T_i, commit>** or **<Ti , abort>**. So, perform **<T_i , start>** 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 **<T_i , commit >**or **<T_i , abort>**, 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 **<T_i , start>**, but in addition it also contains the record **<T_i , commit>** or **<T_i , abort>**. 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 you 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 **<T_i , abort>** 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)

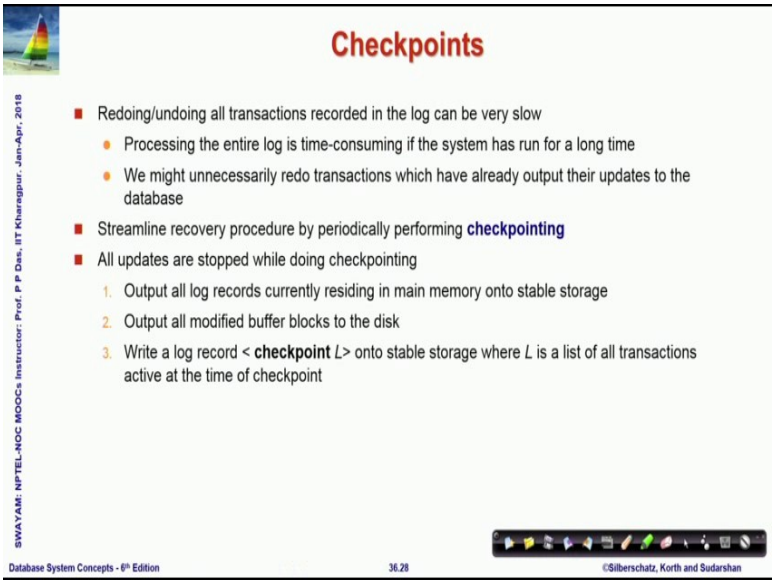


So, here are some examples here they transaction we are showing it is failure recovery action at every case. So, if in case a the transaction has started and we made changes to A and B and at that time the failure happens. So, naturally the start is there and at commit is not there or abort is no there. 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 **<T_0, B, 2000>** and **<T_0, A ,2000>** that two compensation log records will be and then it **<T_0, abort>** 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 **<T_1, C, 100>** and **<T_1, abort>**.

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)



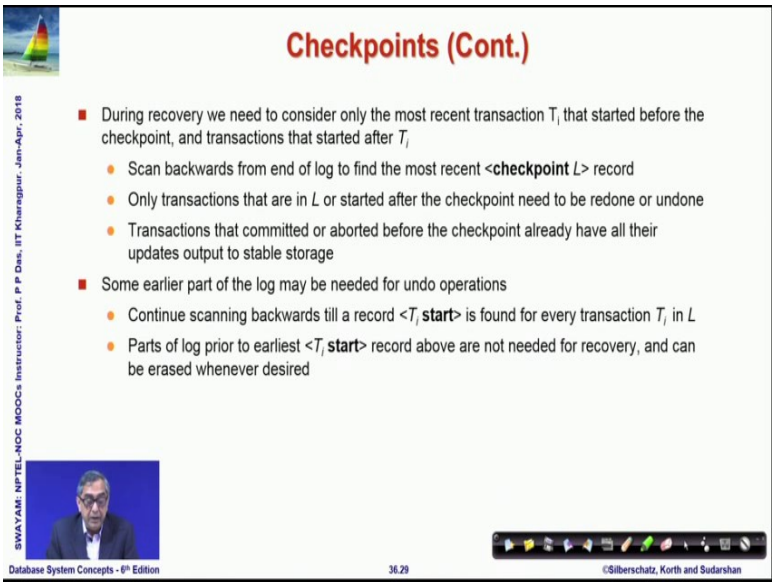
Now, the question is if you have to do this for all transactions when a failure happens and a failure may have happened say after 1 year or after 8, 9, 10 months and so on. So, there will be a huge you know set of redo, undo operations that you will have do it will run for a very long time. So, what 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.

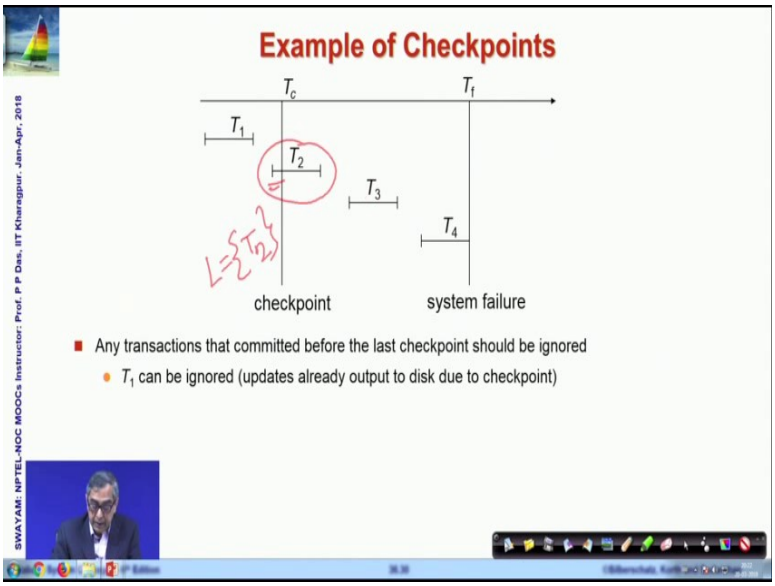
(Refer Slide Time: 36:52)



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.

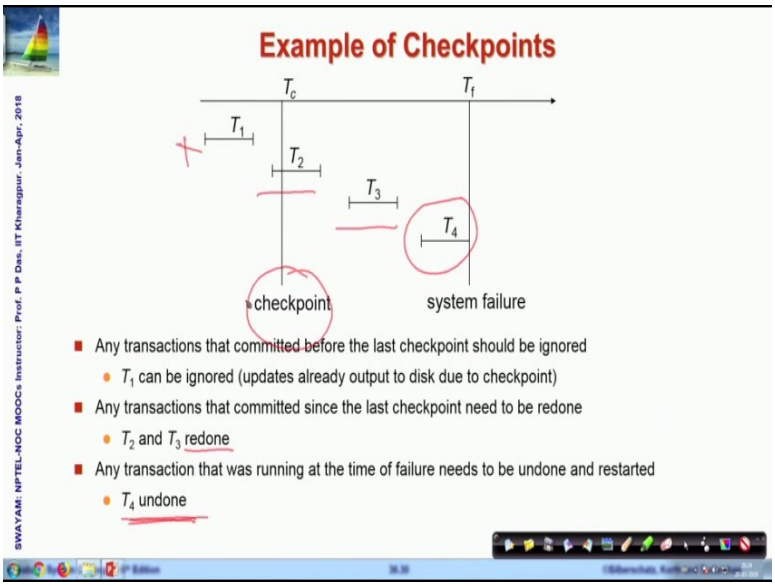
(Refer Slide Time: 37:48)



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 remembering 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)

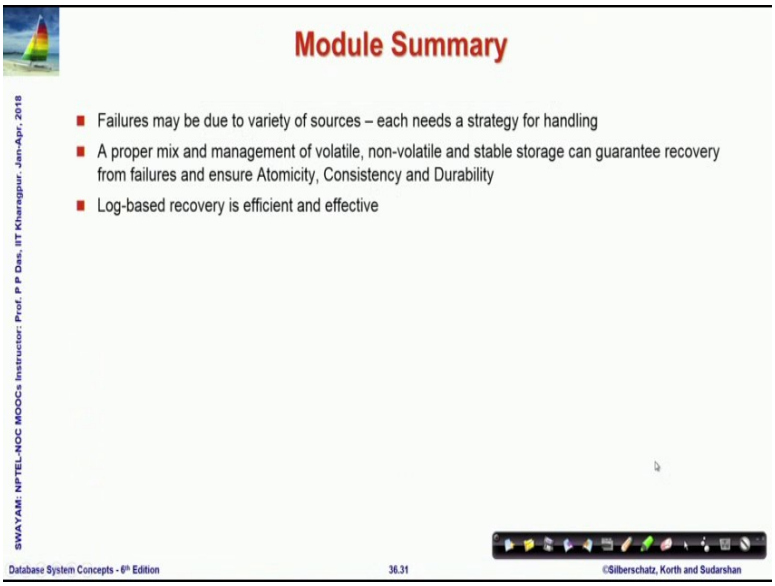


So, if we look at then naturally T_2 if you have a failure at this point if you have a failure at this point as you have here then naturally this is the last stable point you know where everything was written to the disk in a consistent manner. So, what you will need to do you will have to execute transaction 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 the time 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 over it at the same time if you do it in a in a very after 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)



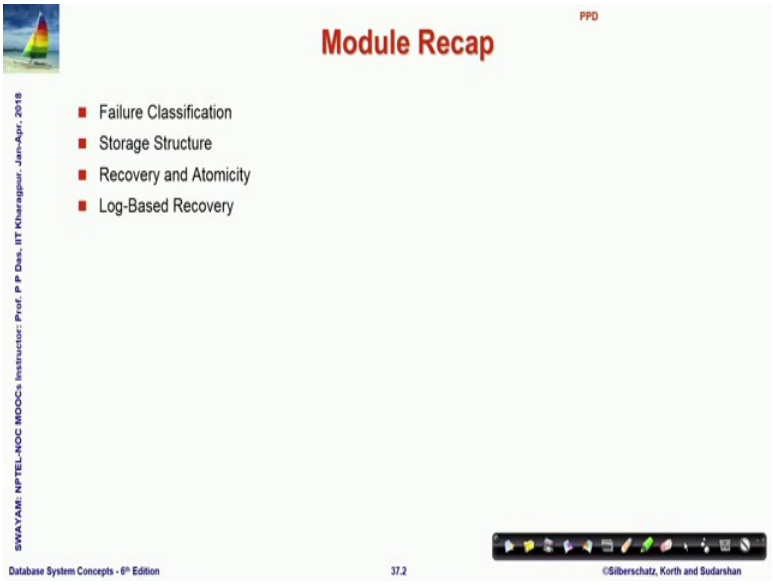
So, to summarize we have seen that there are may be different types of failures and different strategies are required for handling them. And we have also seen that we use different kinds of storage structures and they judicious mix and then arrangement of these the structure can guarantee, recovery from failures. And we have taken their 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

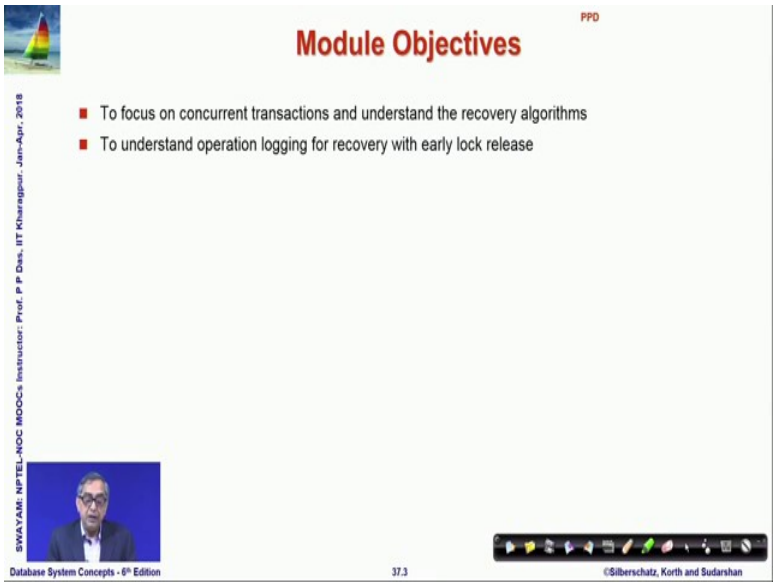
Welcome to module 37 of Database Management Systems. We have been discussing about Database Recovery. This is the second and concluding part of the Database Recovery.

(Refer Slide Time: 00:25)



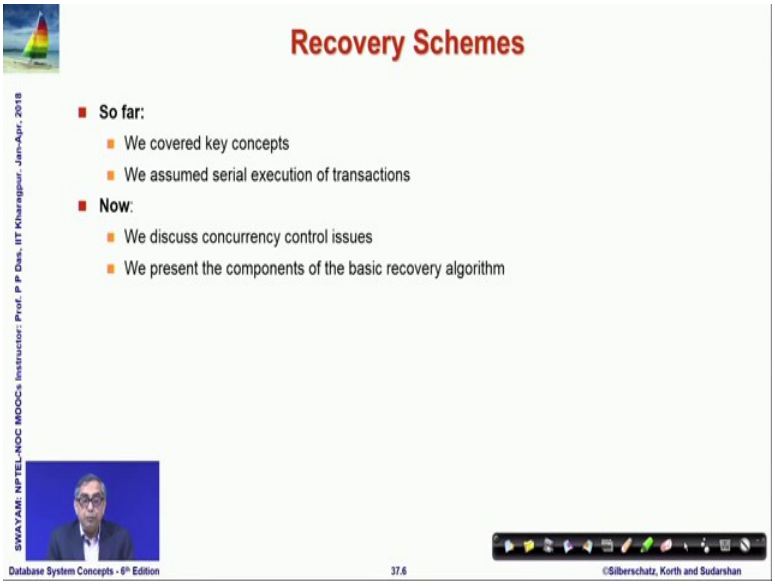
We have earlier discussed about failure classification, storage structures and significantly the log based recovery mechanism.

(Refer Slide Time: 00:34)



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.

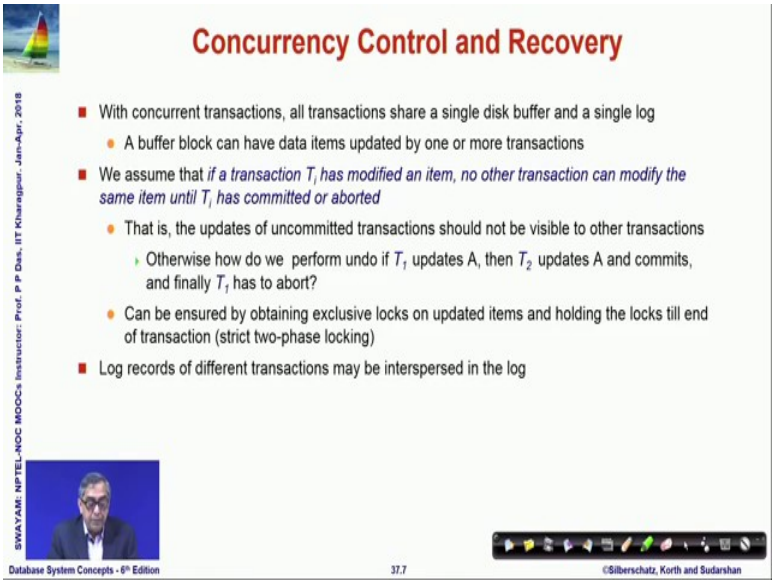
(Refer Slide Time: 00:51)



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)



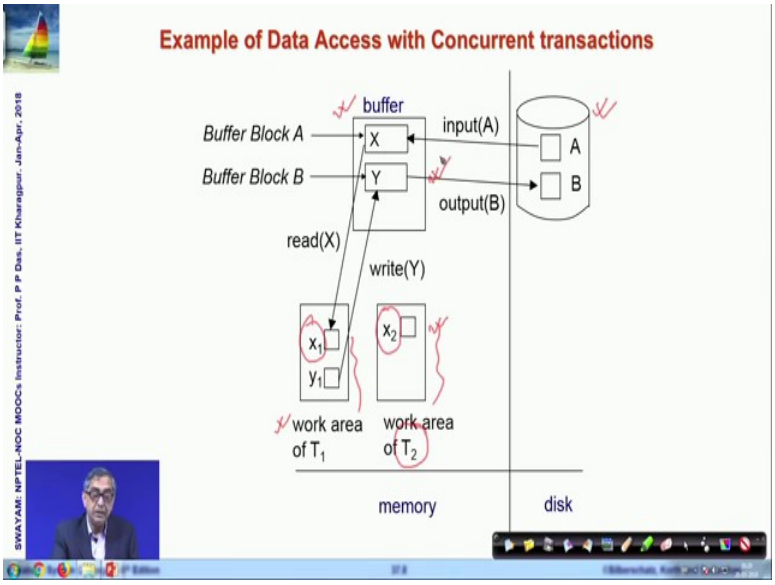
So, with Concurrent Transaction, all transactions share we already know that every transaction is a private work area that assumption stays, but we talked about a system buffer area.

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

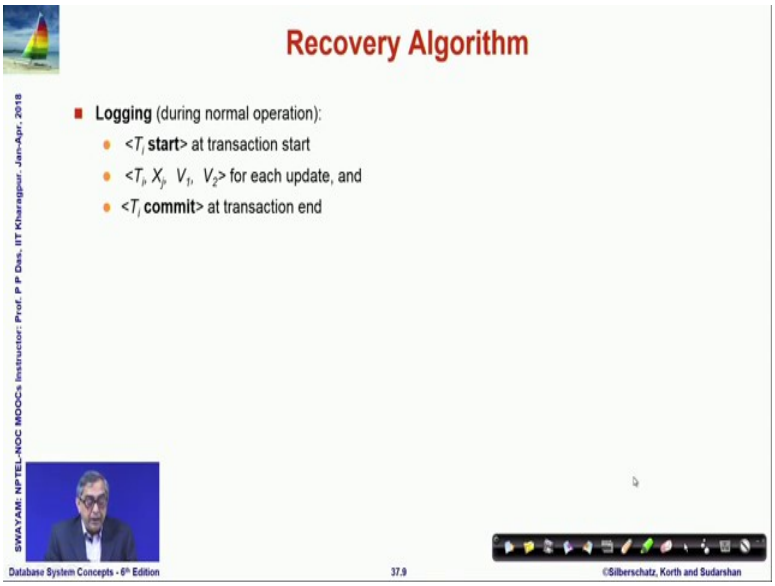
to undo that then we will not know we were which one it to be undone with. So, that is the basic problem. So, we will assume an exclusive lock in this case and log records will be written interspersed as we have already saw.

(Refer Slide Time: 02:43)



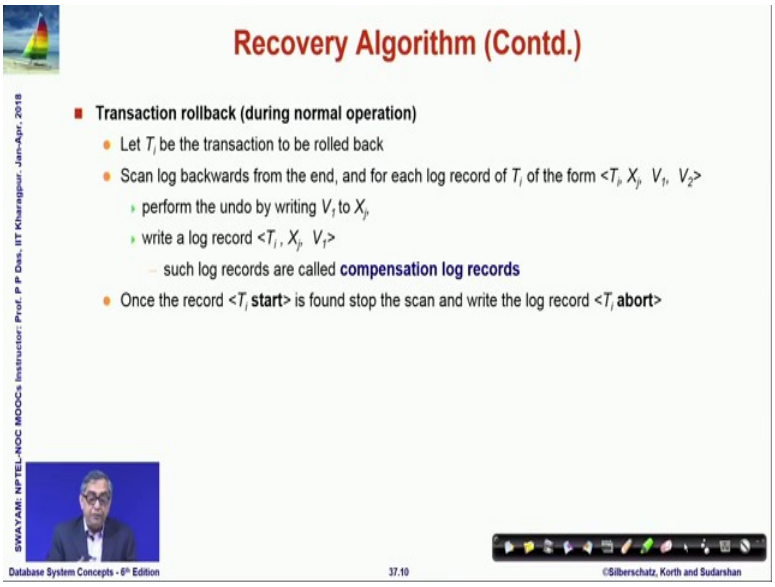
So, in terms of our storage access mechanism, the same eh earlier diagram, this is here is a disk, here is a buffer, the buffer is common and the private work area. So, now, in addition to T_1, we have another transaction T_2 with it is own buffer area, but so, the x has been written in T_1, has been 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 the 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)



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)

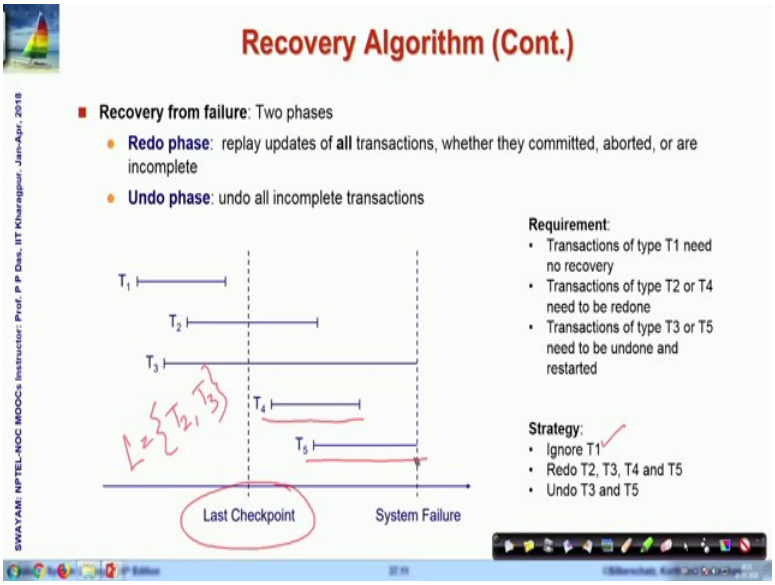


When you have to do a transaction rollback during normal operation; so, for that transaction T i to be rolled back, what we will need to do it is a rollback. So, undo has to happen. So, scan will scan the log backwards from the end and for each log record

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)



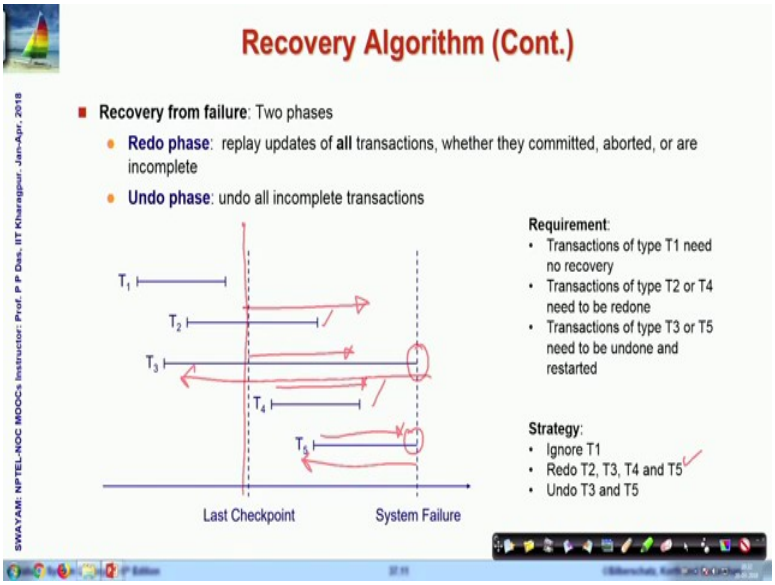
So, now let us look into the actual Recovery Algorithm. So, the transaction rollback has no difference. So, in the Recovery Algorithm, what we do we have a recovery phase and where we replay updates of all transactions. So, we make sure that all transactions whatever they did they those are done again. 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.

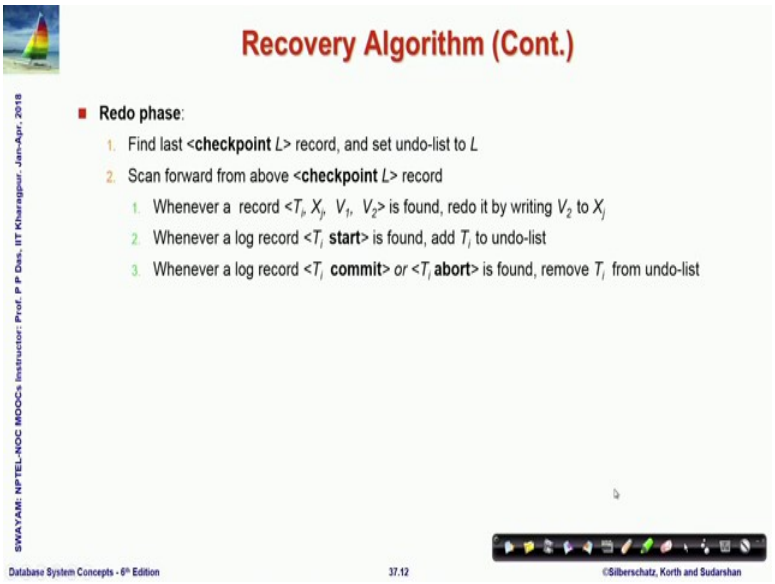
(Refer Slide Time: 06:06)



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.

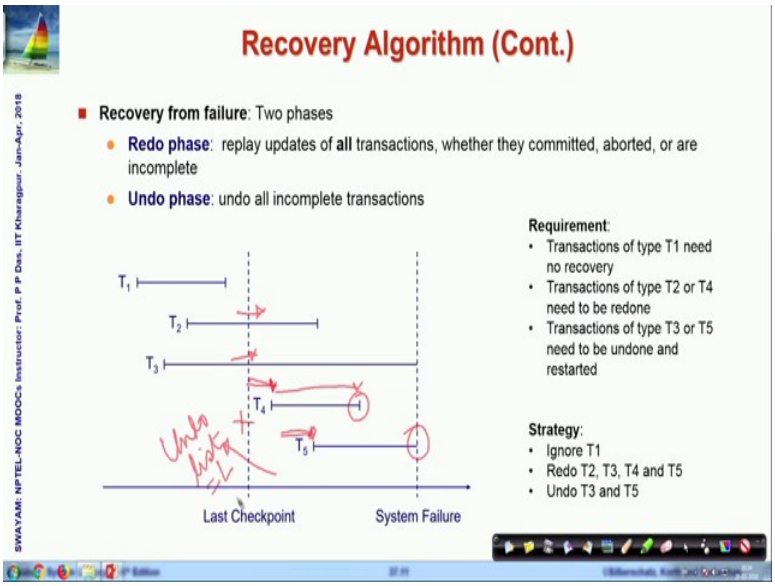
So, after having done this then, we do an undo phase where we undo this, but naturally the effect of these will remain. Now you can question that this could have been done in a more smart way, do we really need to redo everything and then undo some parts of that, that is a override in terms of that which is true, but this just makes the whole algorithm simple and over it actually is not very hard.

(Refer Slide Time: 06:55)



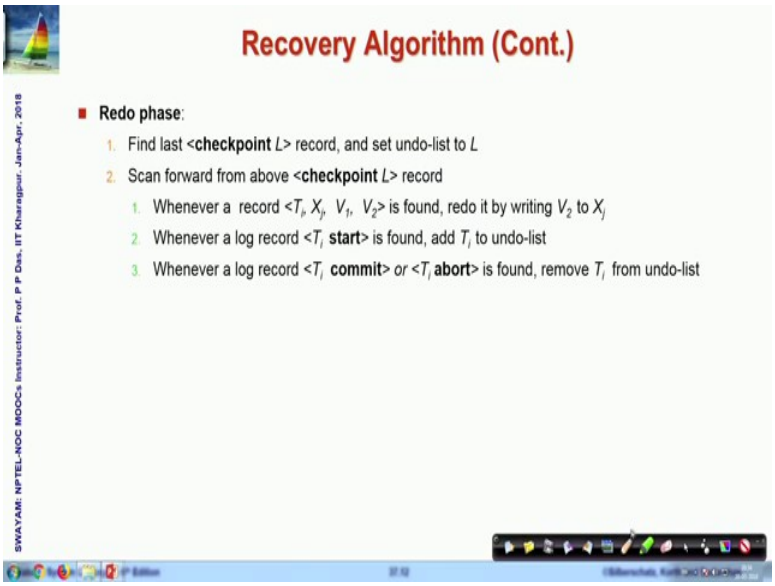
So, we are doing the redo phase, even the redo phase you will find the check point and you will scan forward from the checkpoint record and as you scan forward from the checkpoint record; if you have an update, you will simply redo which means V_2, will again be written to X_j and when you find a start 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)



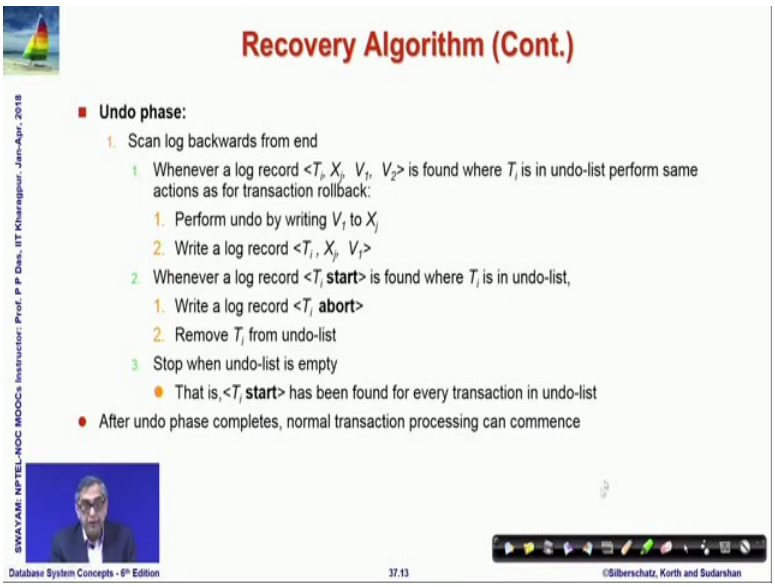
So, whenever you get that, then you put this into the undo list. Initially your undo list is L because they were going on. So, you do not know that they could finish all that still need to be undone and when you come across a new start, you add that to the undo list and then the rest of it is simple. 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)



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.

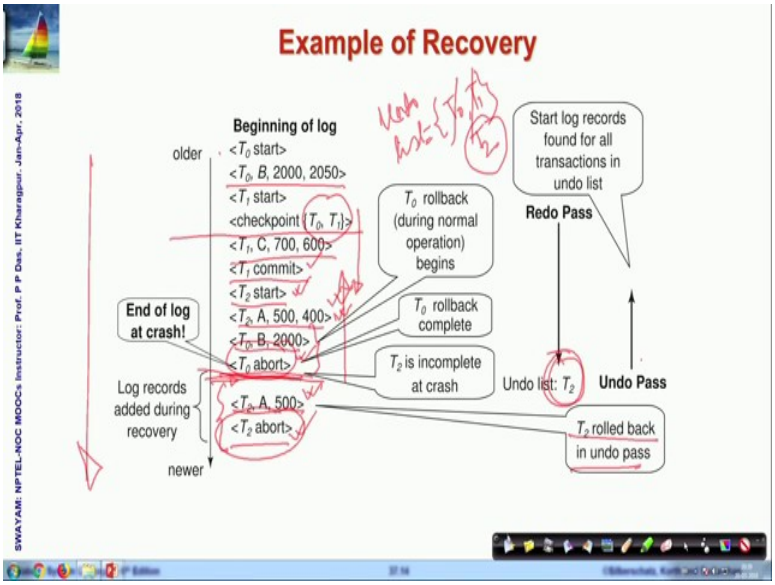
(Refer Slide Time: 08:53)



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 ma 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 **<T_i , start>**; 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 **<T_i , start>** 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

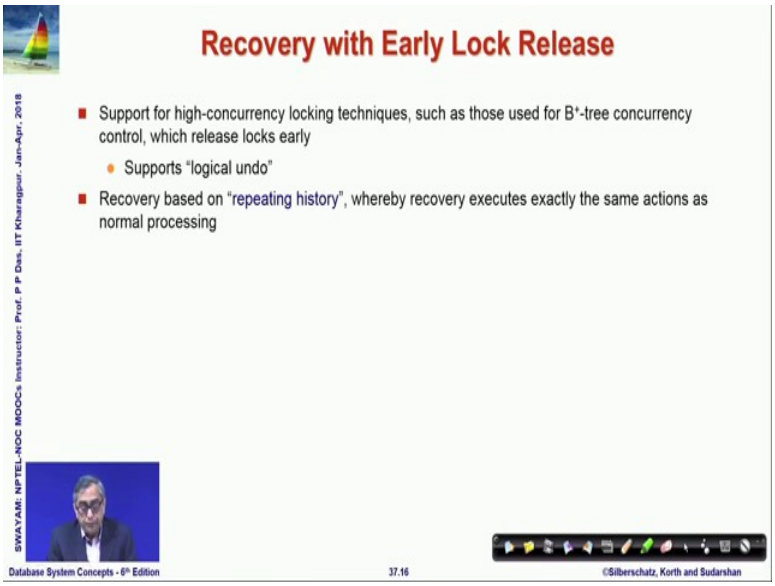
is. So, at checkpoint you can see that T_0 and T_1 are; what are your candidates? So, when you start in the redo phase, you start from this point because before that everything has been done. You naturally, you come across this. So, you redo this which means you again actually change C from it is old 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 undo list at the beginning is T_0, T_1, but going in the forward direction when you come across **<T_1, commit>**, you naturally that from your undo list, then you come across **<T_2 , start>**. So, you know that another transaction is starting now. So, it may be you do not know whether it come 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 in 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, this 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 can be rolled back in the undo 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 ma we would like to talk about is Recovery with Early Lock Release.

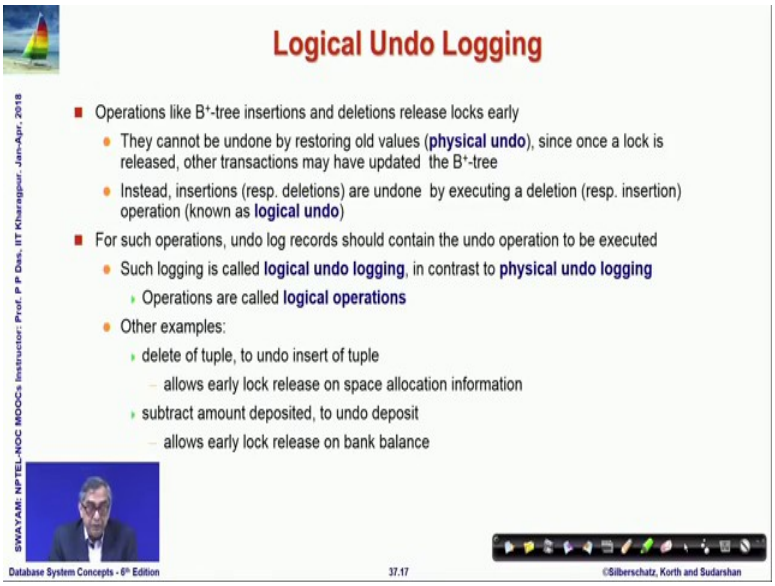
(Refer Slide Time: 13:55)



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.

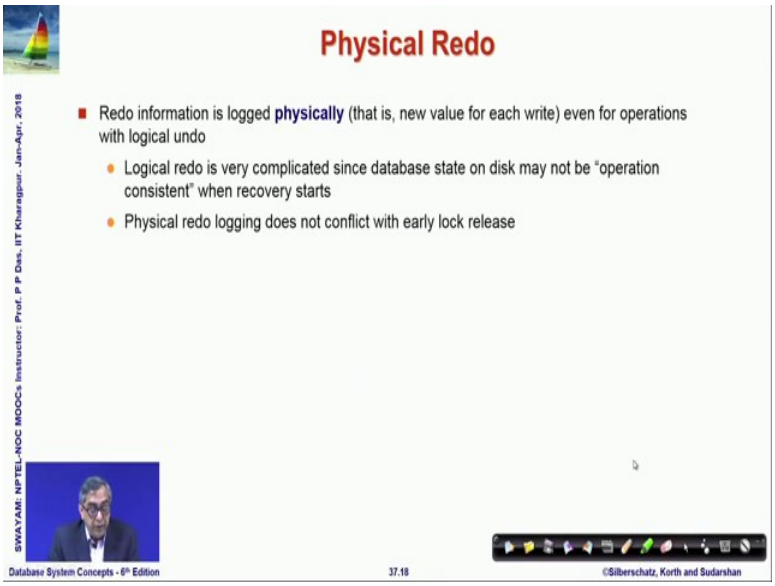
(Refer Slide Time: 14:53)



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 you 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.

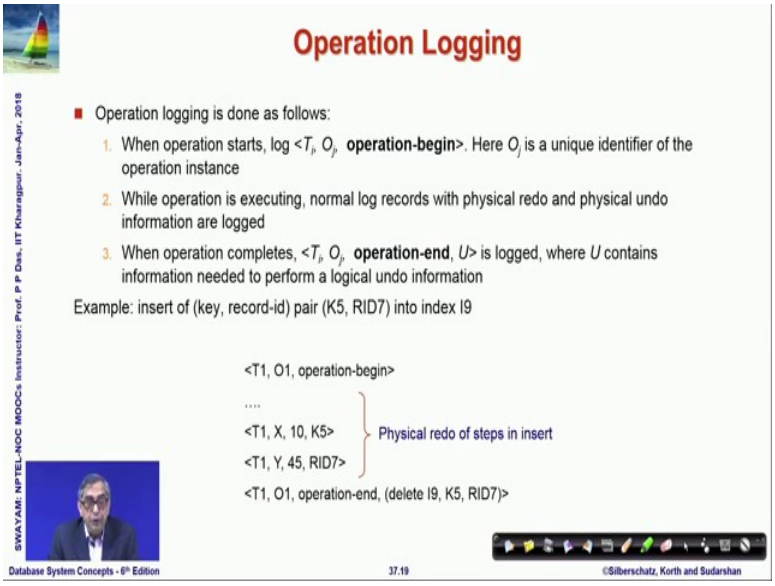
So, if you have deleted a people undo to insert, if you have subtracted then undo to undo deposit to go forward and so on.

(Refer Slide Time: 15:55)



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)



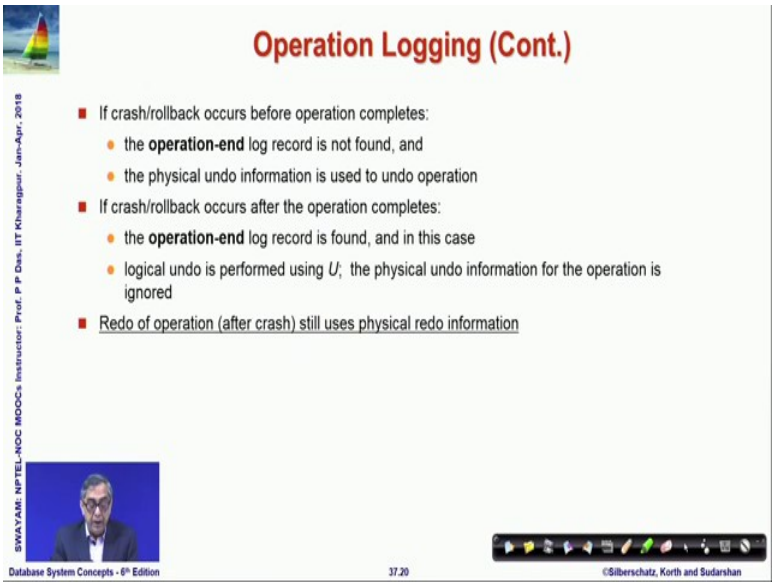
So, how do you log for such a logical undo operation, what you do is instead of now. So, now, it is an operation it may not be an a single value update. So, it is not captured in terms of one you know log record, but it could be a number of log records which have actually done three, four different changes 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.

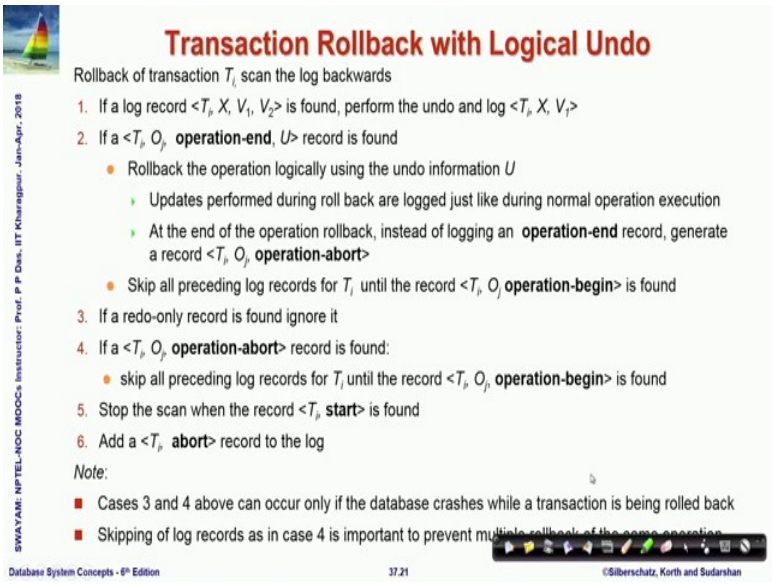
So, now you do write your what will be the undo, to delete that from index I9, to delete this key record ID pair. So, this is a whole locking that we do. So, you can make use of this undo operation in terms of your recovery process.

(Refer Slide Time: 19:22)



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.

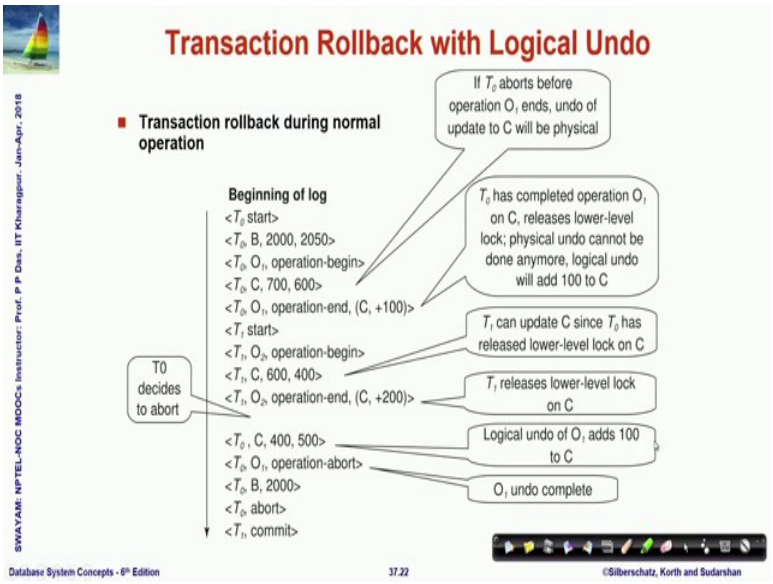
(Refer Slide Time: 20:10)



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 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 **<Ti , abort>** 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.

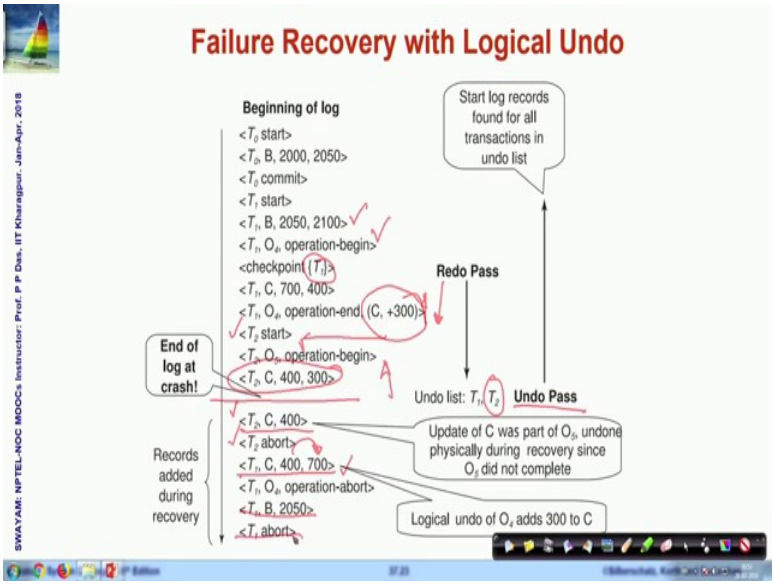
So, if T_0 aborts here, if T_0 aborts somewhere here you know before your operation end has happened then naturally the undo will have to be based on this physical undo structure. So, you will have to replace 600 by 700, but if it happens after this, then this is the case if it is completed the operation 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 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)

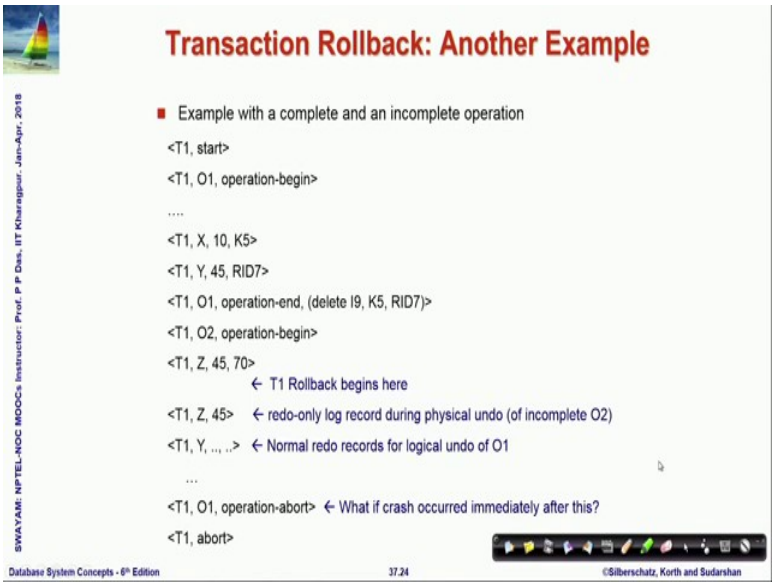


This is similarly another illustration for doing the failure recovery for with logical undo. So, here is the undo is a re redo phase that you are seeing here, this is where the end of log at the crash these are redo phase because these are check point where T_1 was there. So, at the end of redo T_1 if 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 T_1 had started.

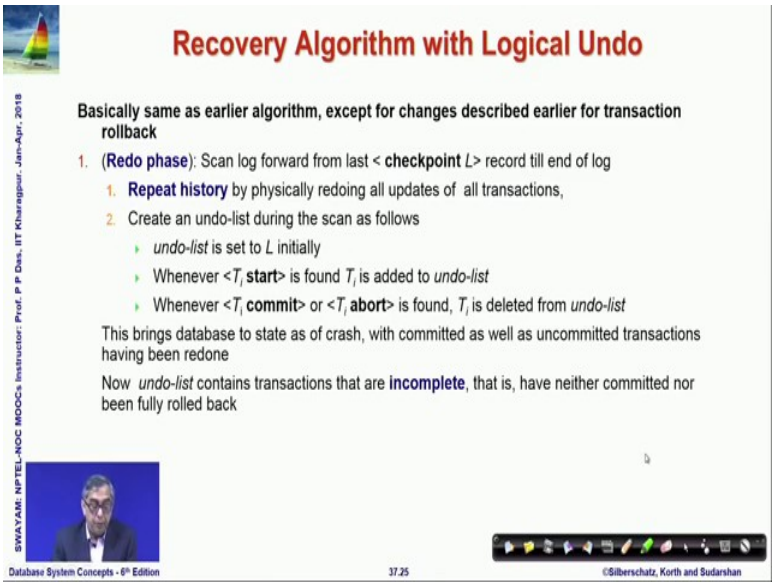
So, you undo that. That is again a physical undo and finally, T_1 aborts. So, this is how in both cases of transaction rollback as well as in terms of the failure the recovery can be done with the logical undo process.

(Refer Slide Time: 27:10)



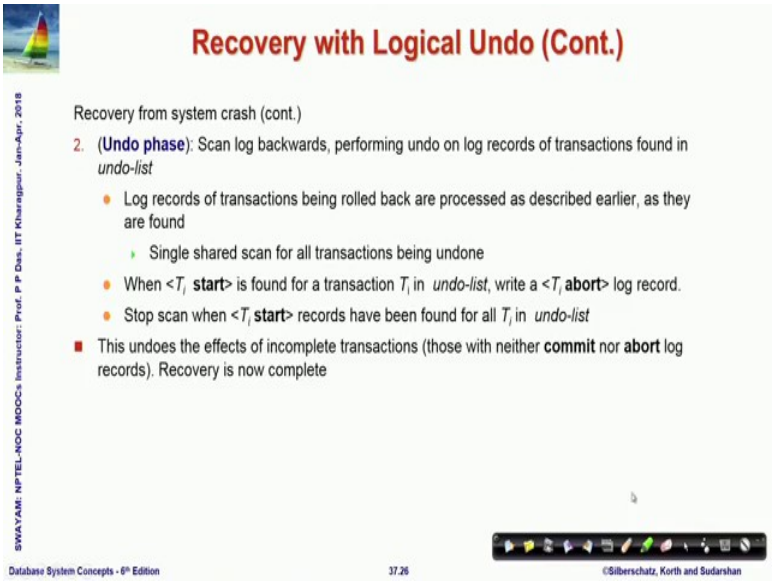
Here I have given another example. I will not go through it step by step. So, at a arts that you go through that following the same logic and convince yourself that you understand that how this transaction rollbacks with physical undo as well as logical undo is taking place.

(Refer Slide Time: 27:27)



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.

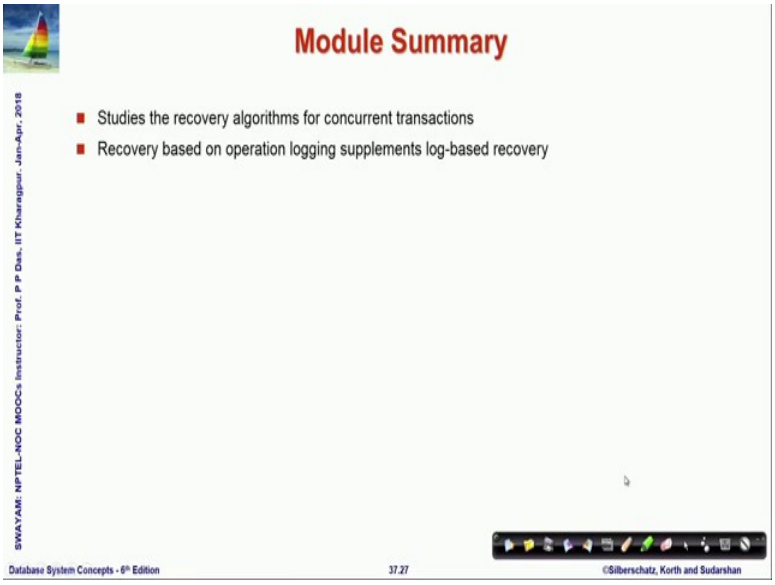
(Refer Slide Time: 27:52)



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)



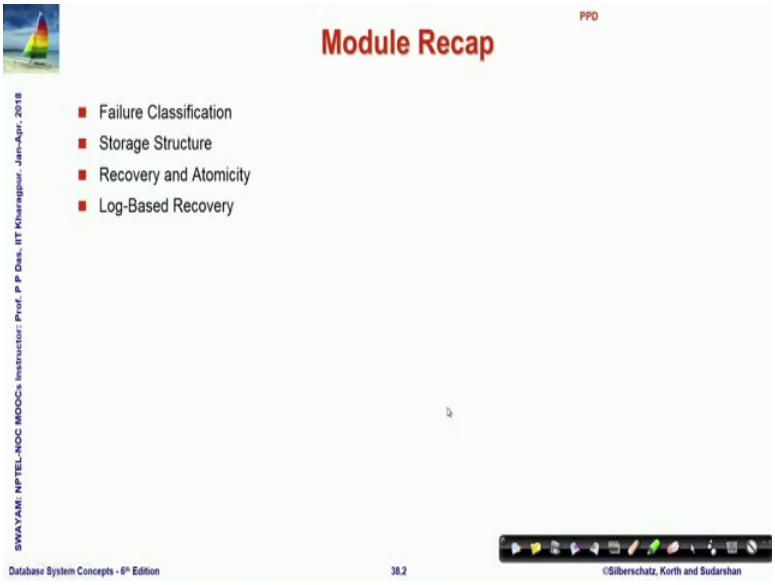
So, in this module we have expose ourselves with the Recovery Algorithms now for concurrent transactions as well and we have shown that how recovery can be done using operational logging, operations logging and making sure that really the database may not need to hold on to a lock for a long time on the data item and delay other transactions, but if it can define the undo operation on the on the data 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

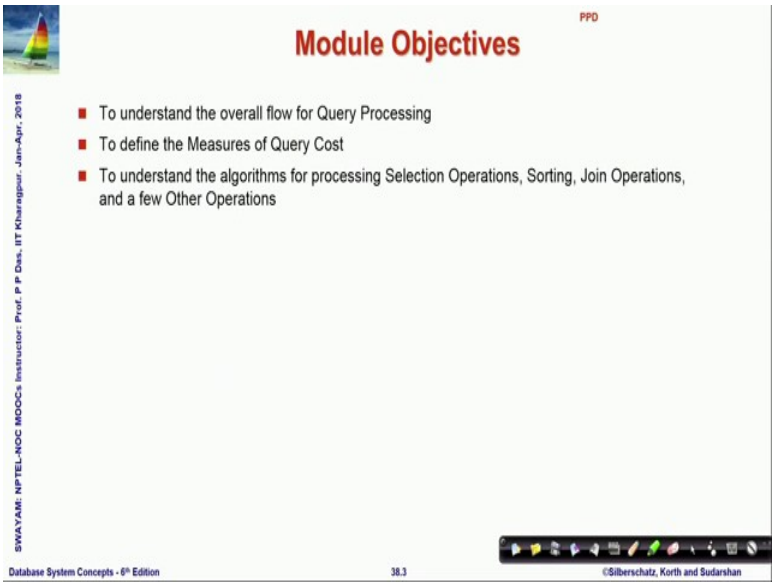
Welcome to module 38 of database management systems, in this module and the next we will talk about query processing and optimization of that in the current module we will talk about query processing.

(Refer Slide Time: 00:29)



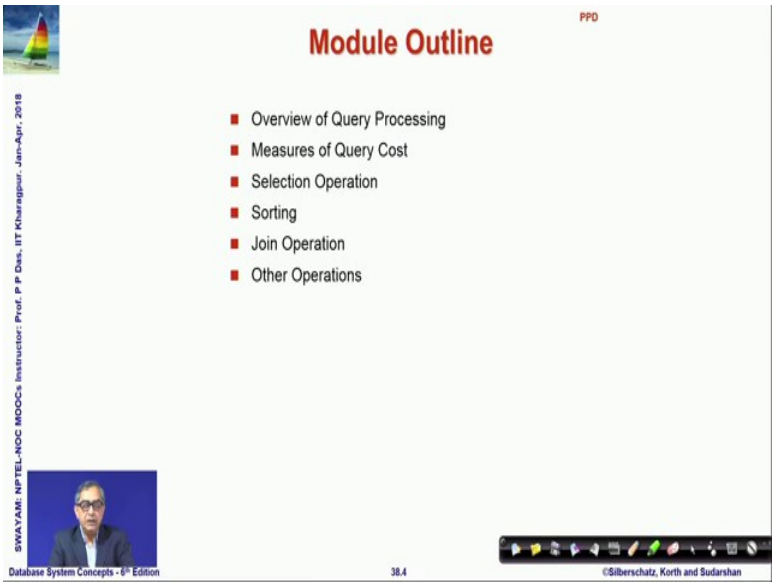
So, in the last module we had done talked about database recovery.

(Refer Slide Time: 00:36)



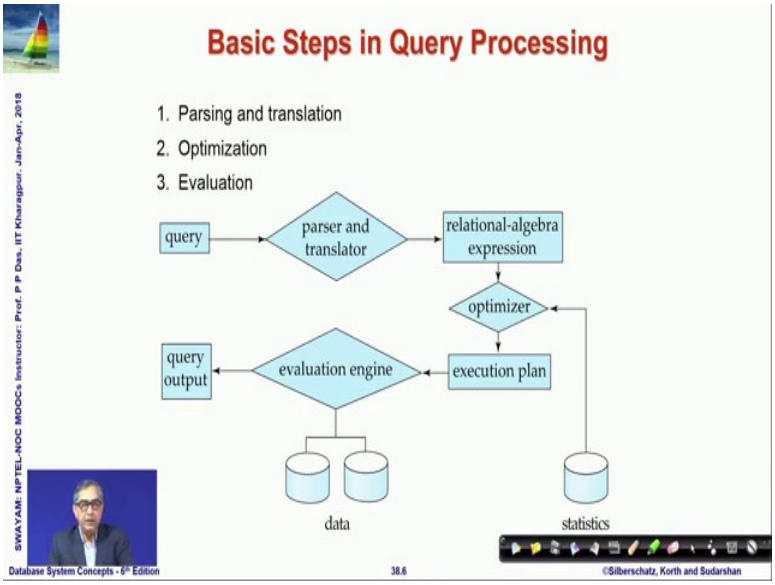
And now we will try to understand the overall flow of processing queries. So, if I fire a query like select from where how will that actually access the database files the B trees and indexes and so, on and compute the result is what we would like to discuss. And a query can be processed in multiple 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)



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.

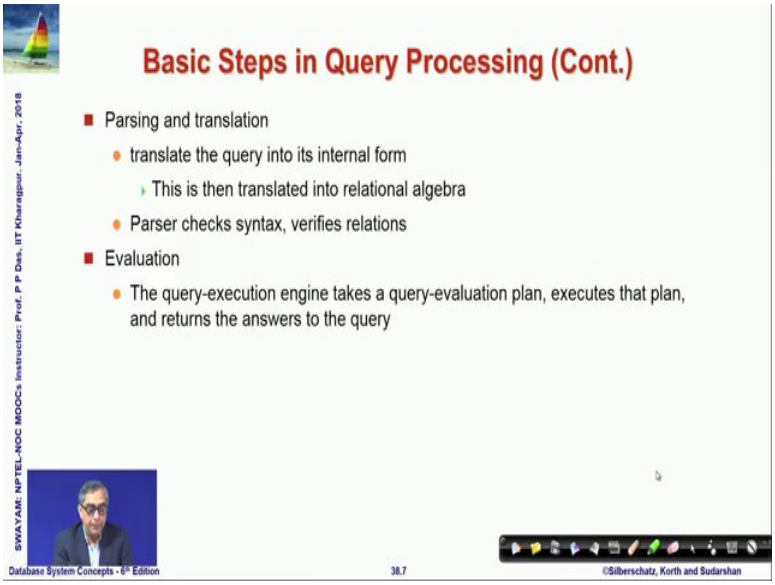
So, you need a parser and translator. So, the it is parse translated and it is translated into a relational algebra expression as we have shown at the very beginning discussed at the very beginning that SQL is basically derived or developed based on relational algebra. So, corresponding to every SQL 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.

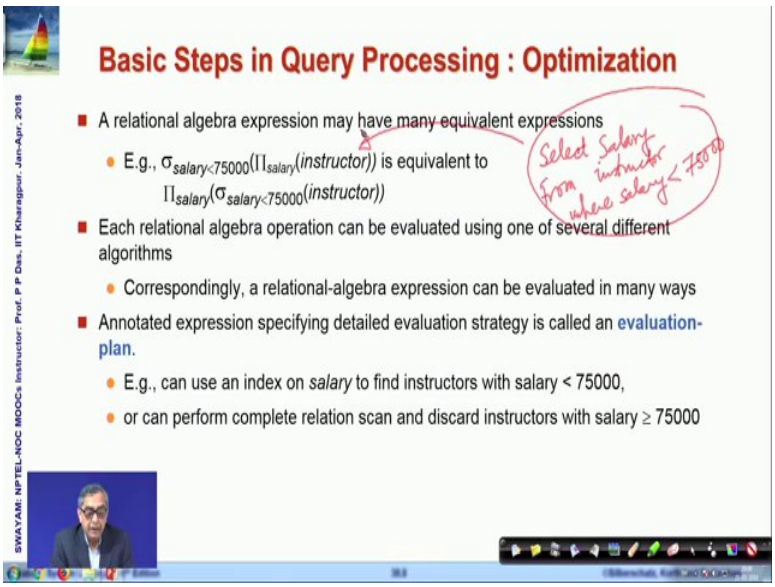
So, in this module and the next we will take a quick look into so, it is glimpses of these steps one by one and try to understand how query processing and optimization can happen.

(Refer Slide Time: 03:40)



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

(Refer Slide Time: 03:48)



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**.

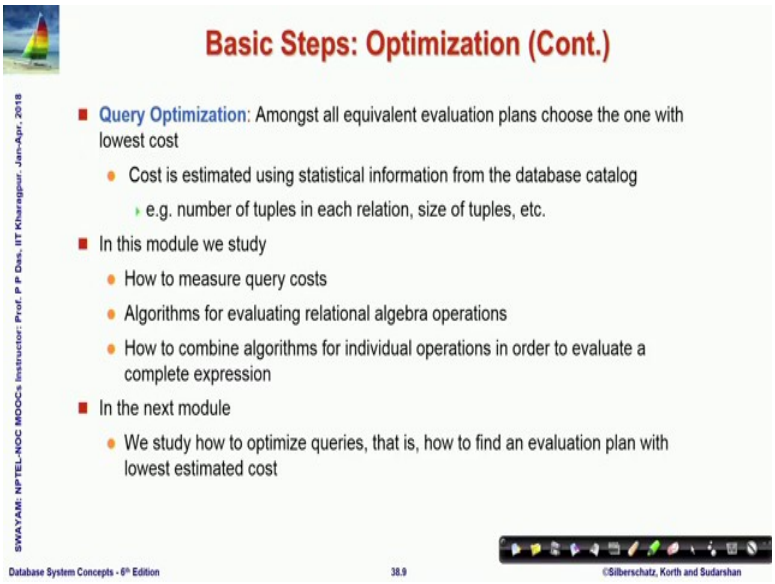
So, if you had a query like this then you know then it will be it will get translated to some kind of a relational algebra expression like this where you do a selection on the salary and then you do you do a projection on the salary and you do a selection based on that condition. Now, it is also clear 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)

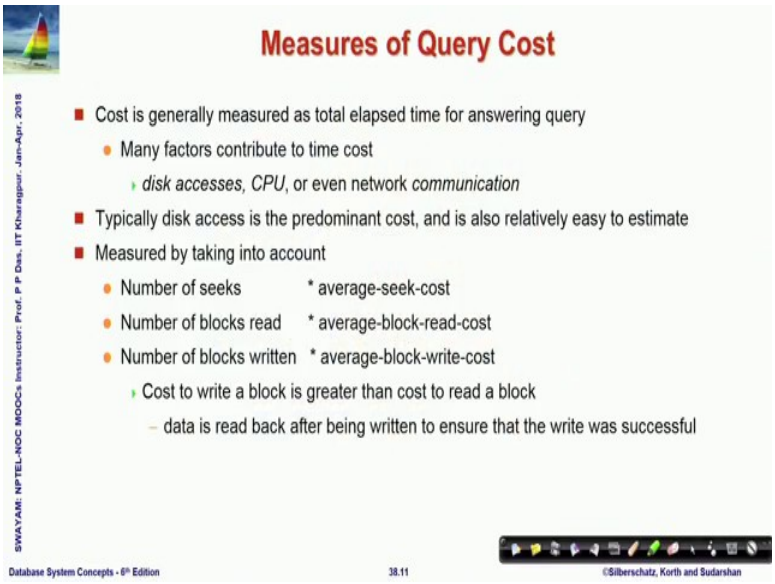


So, in terms of query optimization out of all these equivalent evaluation plans we try to choose the one that gives some minimum cost the lowest cost evaluation.

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)

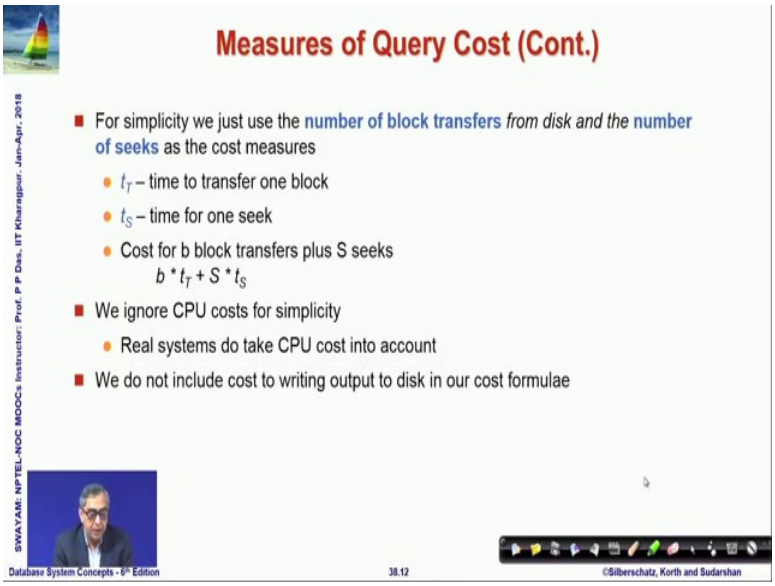


So, measures of cost will be in absolute terms it is in terms of the elapsed time how much time does it take and there could be many factors which ma dictate that because in terms of evaluating this we will have to access the disk. So, access time of the disk will be involved, the computing time in CPU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

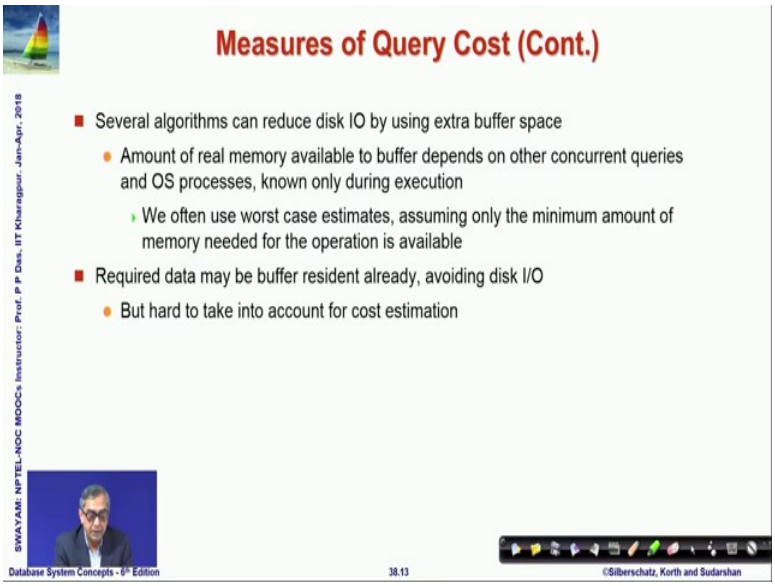
(Refer Slide Time: 09:53)



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)

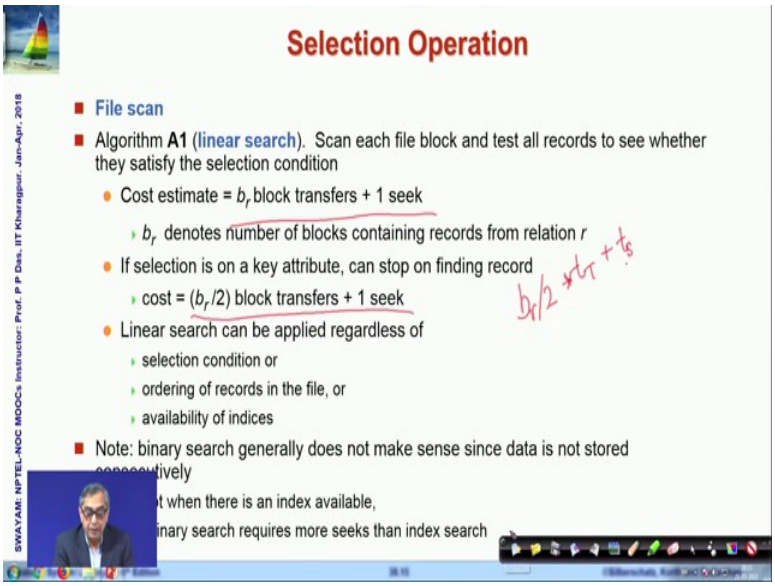


So, there are also it has to be noted that there are also several algorithms to reduce the disk I/O we can do that by using extra buffer space for example, one block has been read in and we are just using one record of that if in the next operation we have to use some record which is already existing 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)



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

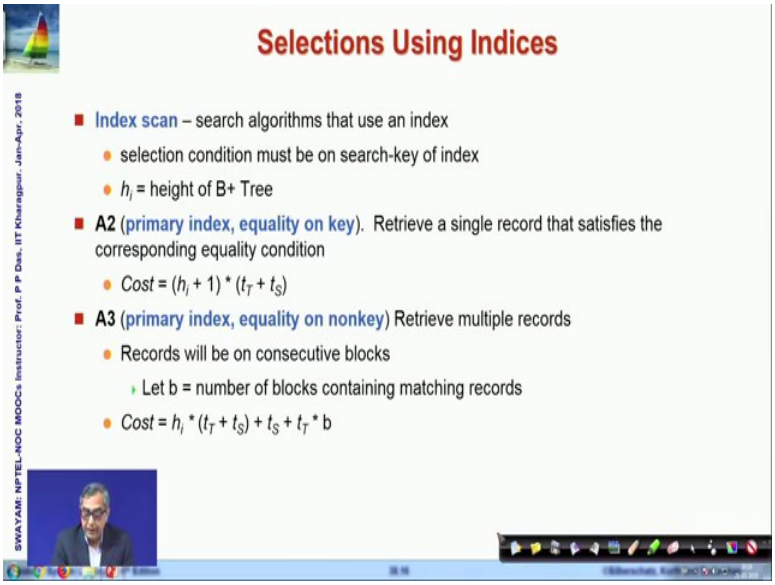
So, the cost for that would be b r block transfers if there are if b_r is a number of blocks containing records from relation r then b_r blocks have to be read and one seek has to happen. Now, if the selection is on a key attribute and we can stop find on finding the record and on the average we can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



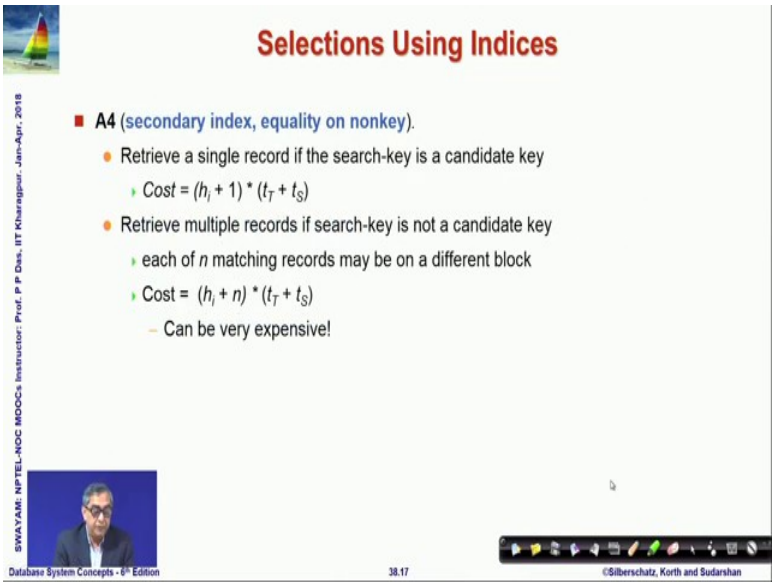
Now, if now let us assume that it is the situation is such that we have some index on the B tree B + tree that we are using to going to do the selection on and let us assume that h_i is the height of that B+ tree. So, the second algorithm is good if we are using a primary index and we are looking for 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 hi 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)

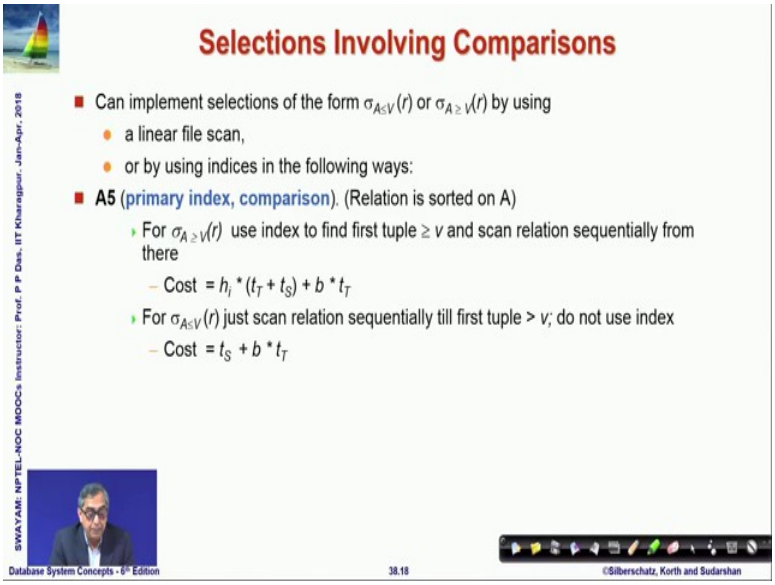


If we look into a few of the other situations for example let us say instead of primary index if I have a secondary index and you are looking for a equality or non key then you can retrieve a single record if the search key is candidate key. If it is a candidate key then we know that even though it is 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)

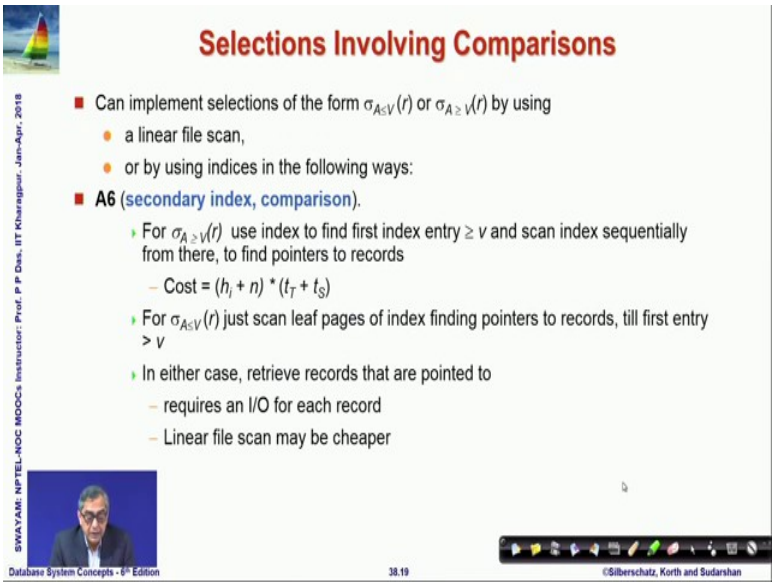


We can implement different this is a very common selection condition where we have these kind of conditions that we are selecting on less than equal to or greater than equal to kind of condition.

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.

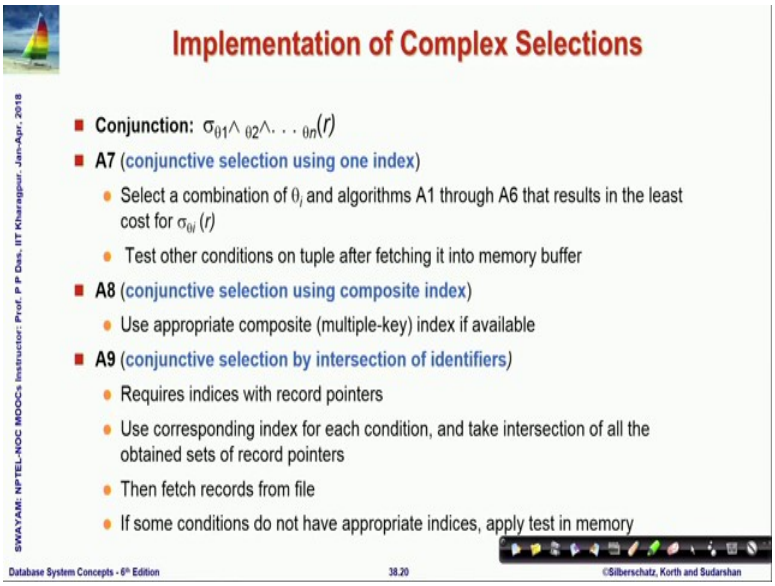
(Refer Slide Time: 20:03)



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

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

(Refer Slide Time: 20:59)

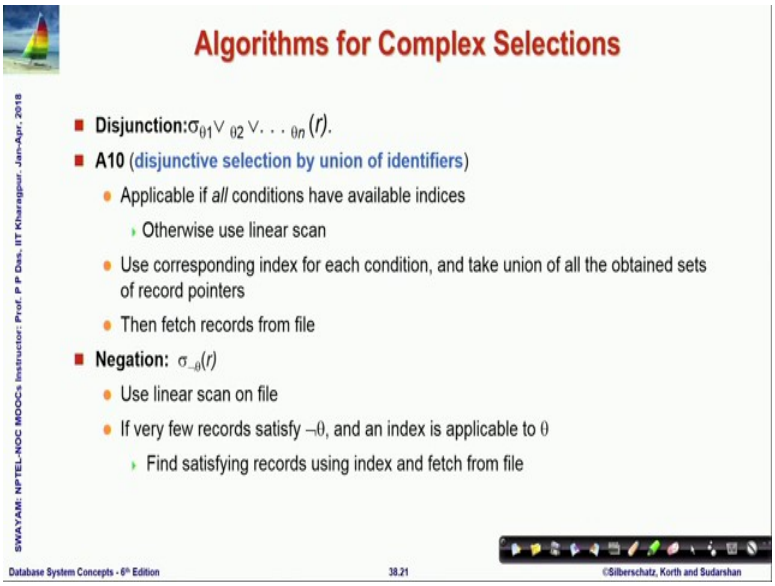


Often we have select conditions which are conjunction. So, it could be conjunctive select and may be using only one index in that case it depends on if there are n conditions then we will need to depending on the combination of this condition Θ_n and the algorithms that we have seen here we can evaluate 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 Θ_1, Θ_2, Θn 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.

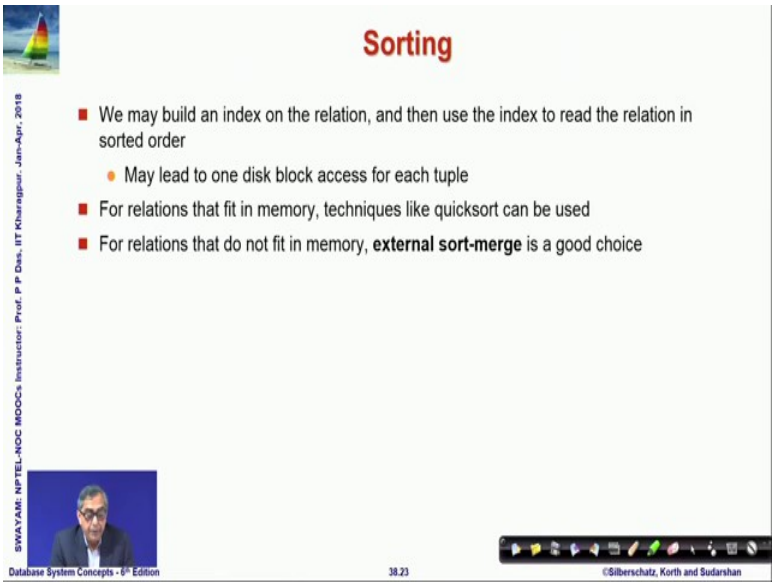
(Refer Slide Time: 22:29)



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 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.

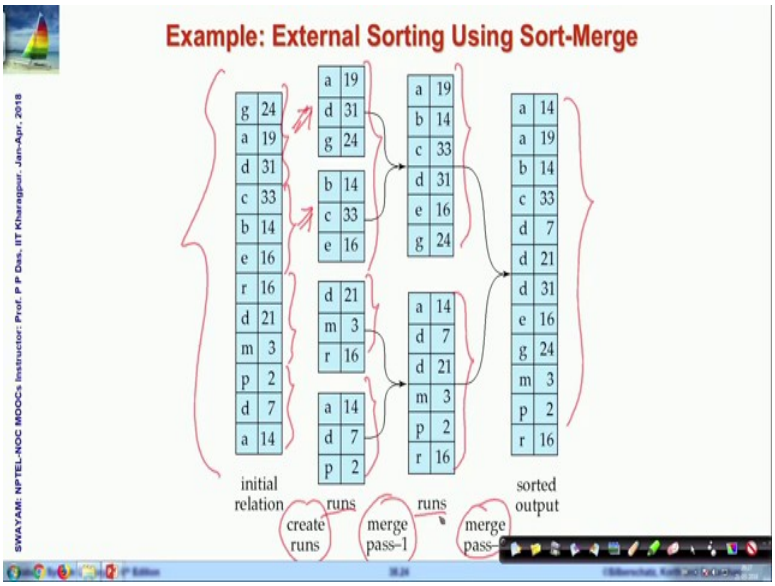
(Refer Slide Time: 23:47)



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 if all the records can totally fit into the memory then we can use some in memory algorithm like quicksort, but often that will not be the case relations are much bigger.

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

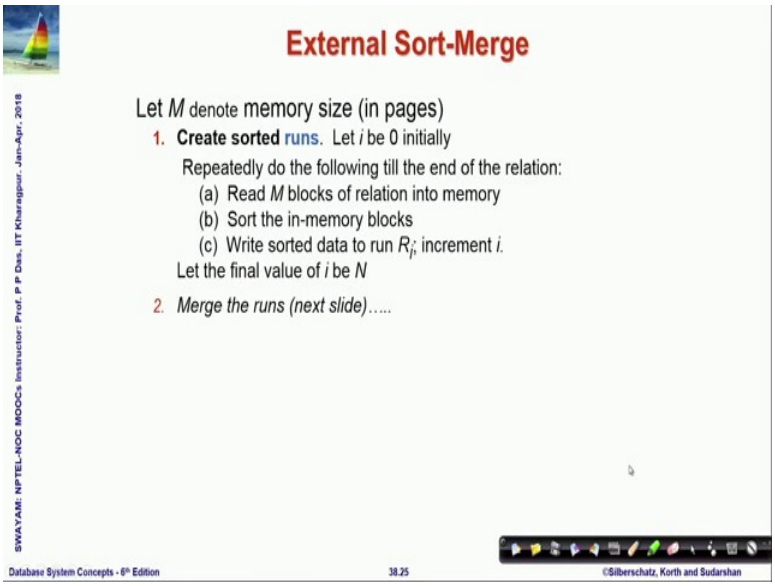
(Refer Slide Time: 24:30)



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

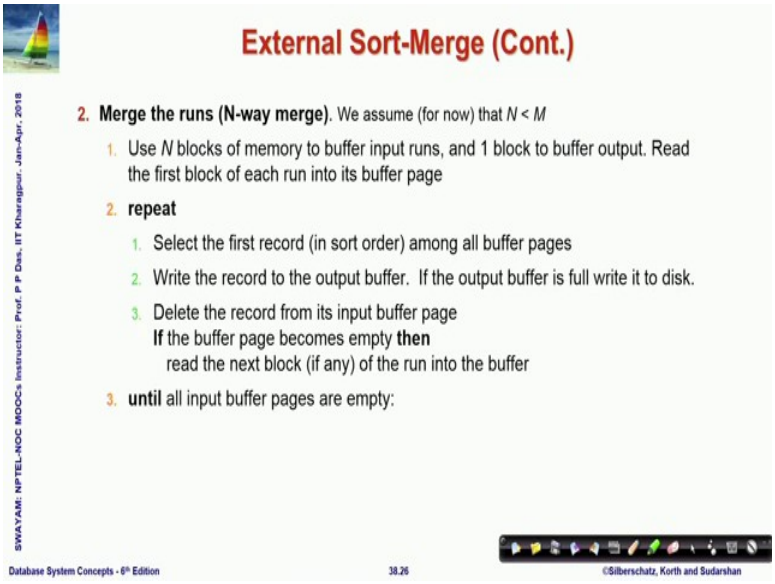
So, once you have sorted them then you can these are two sorted sub lists of the original set of records. So, now, you can merge them according to the merge strategy so, this is the sample merge saw strategy and again you write this back you do the similar things again here write them back. So, now, 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)



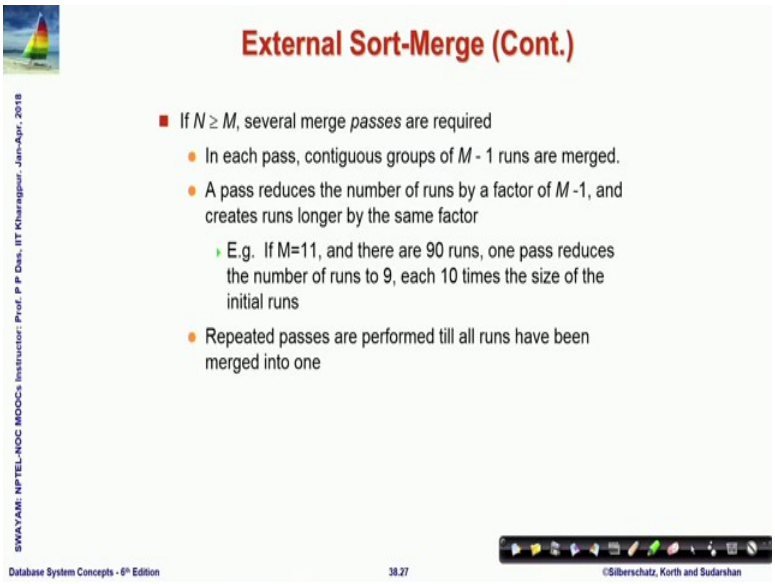
So, that is a very external sort merge is a very effective strategy that the databases will always use and the efficiency of that or the cost of that depends on the size of the memory in terms of pages as to what can fit a complete one run data. So, this is whatever I have described is simply given here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

(Refer Slide Time: 26:16)



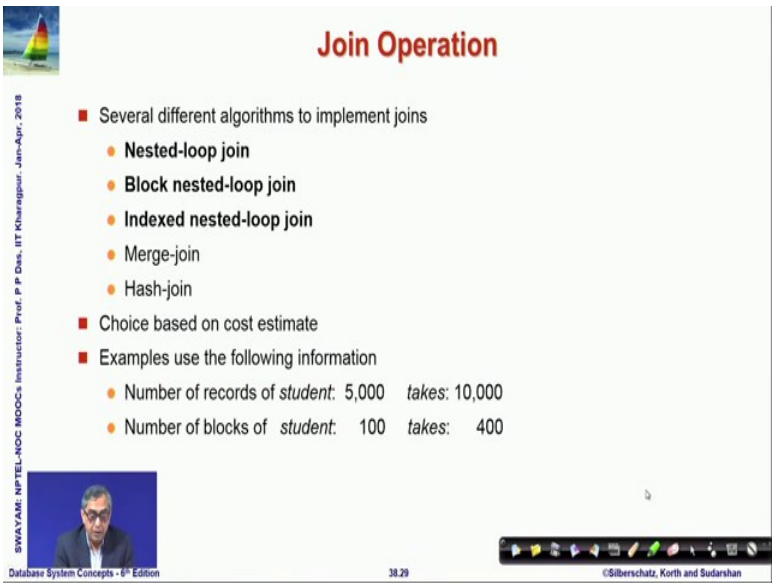
So, that is a sort that is that here is a merge so, you can go through that and convince yourself that this is what algorithm is actually doing.

(Refer Slide Time: 26:24)



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.

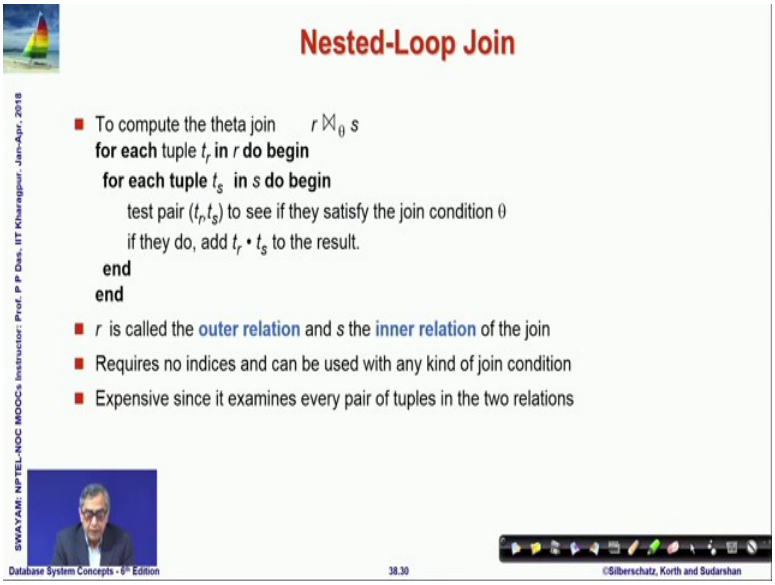
(Refer Slide Time: 26:56)



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.

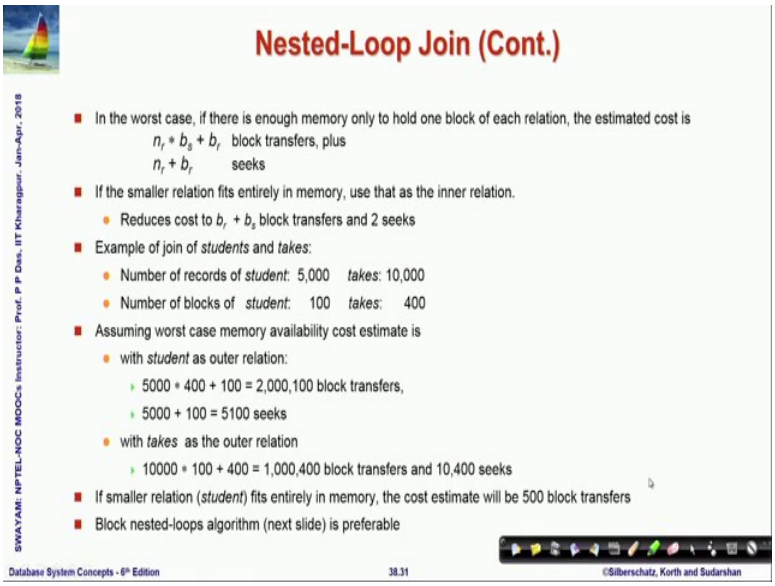
(Refer Slide Time: 27:15)



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 n 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)



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.

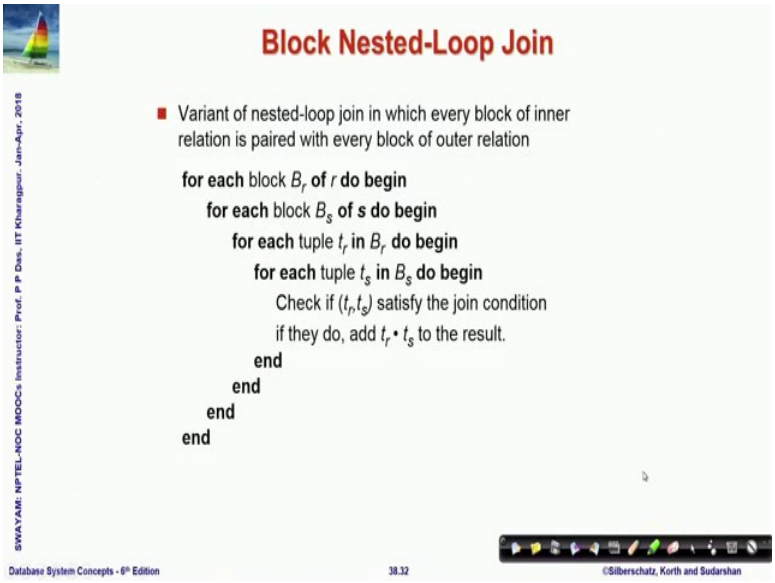
So, here I have just shown a simple example of computing the join of student and takes let us say student has 5000 records and spread over 100 blocks takes relation has 10000 records spread over 400 blocks then if you apply the formula above you will find that if student is the outer relation you have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

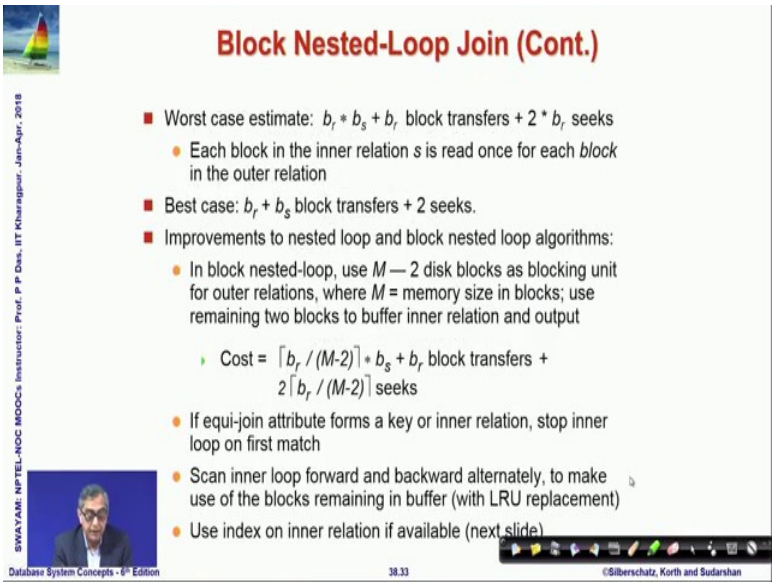
So, will possibly in with this kind of a statistics if it is available then we will possibly take takes as outer relation and student as the inner one. You can refine this.

(Refer Slide Time: 31:07)



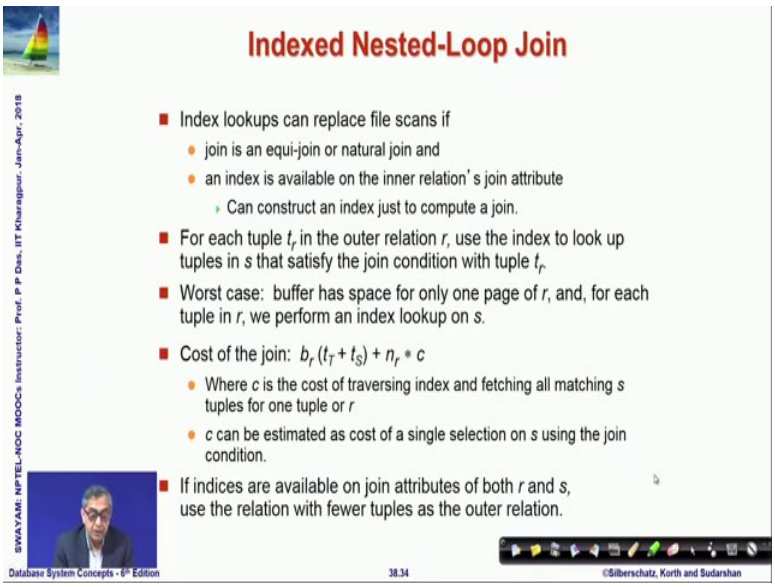
Strategy by doing a block nesting that is instead of taking every tuple of the relation you can take every block of the relation. So, for every block of relation r you try to match with you try to combine with every block of relation s and then within every block of relation r the block b_r you take 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)



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)



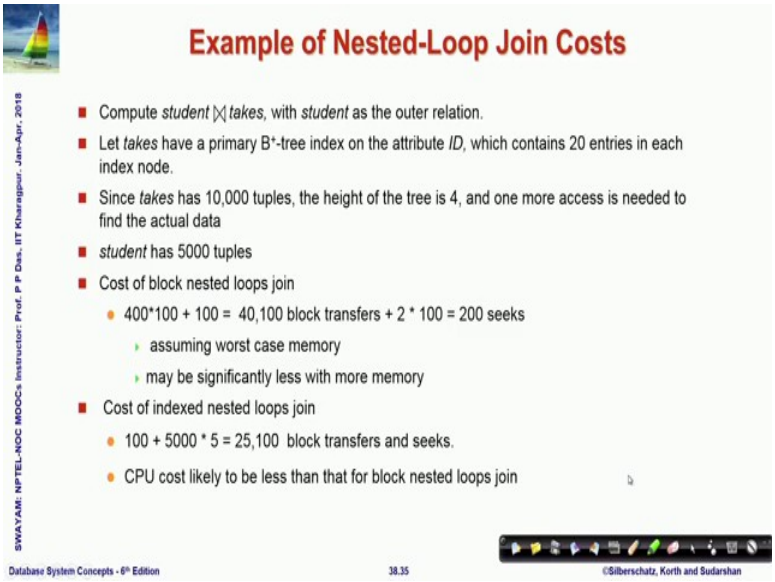
The third strategy which we will use very often is efficiently applicable if you are if your join is an equijoin or a natural join as we have seen that we often need to do a natural

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.

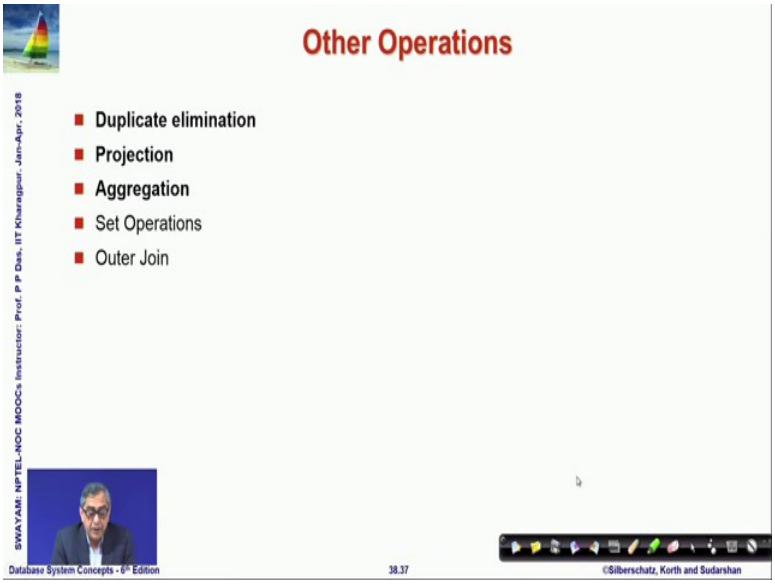
(Refer Slide Time: 33:44)



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

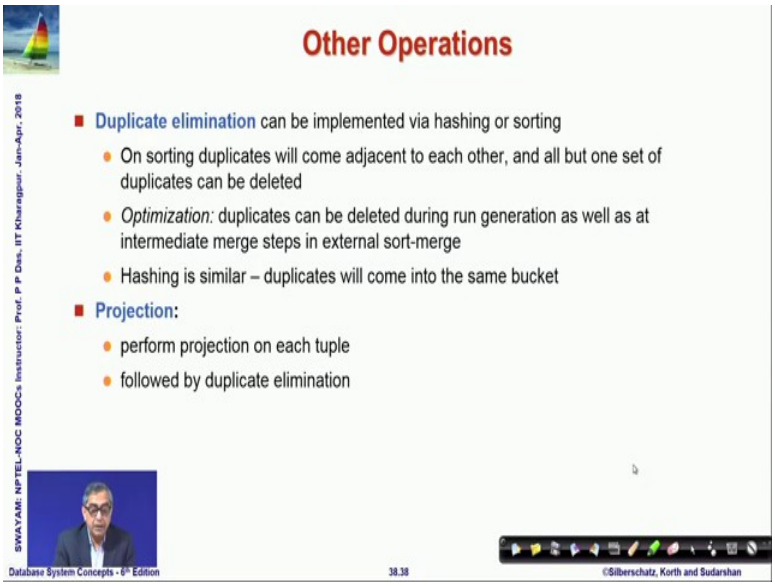
So, these are there are several other strategies particularly hashing based strategies, merging based strategies which we are not discussing here, but there are different strategies through which you can do join in more and more efficient manner.

(Refer Slide Time: 34:36)



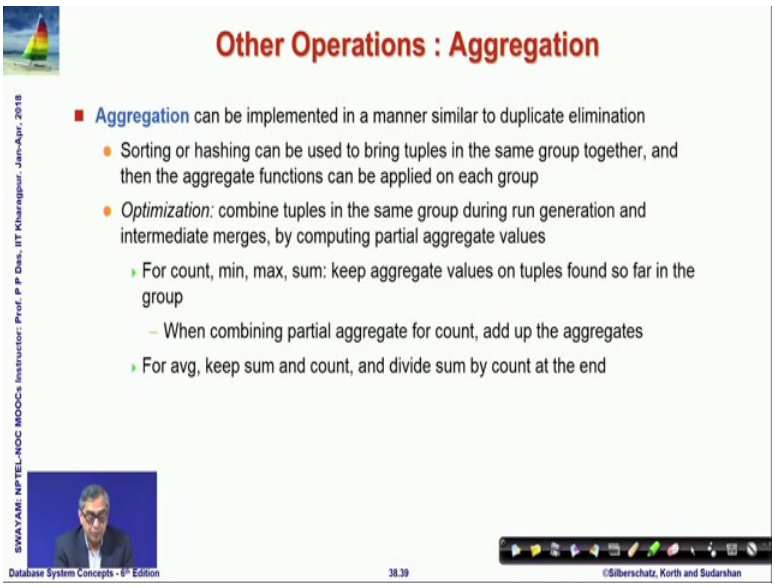
Couple of other operations which are often required is duplicate elimination because if they we know that there are duplicate records cannot be kept, duplicate in the sense the records which match in the in the key field and the duplicate will often happen in terms of the result, they will happen in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

(Refer Slide Time: 35:05)



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

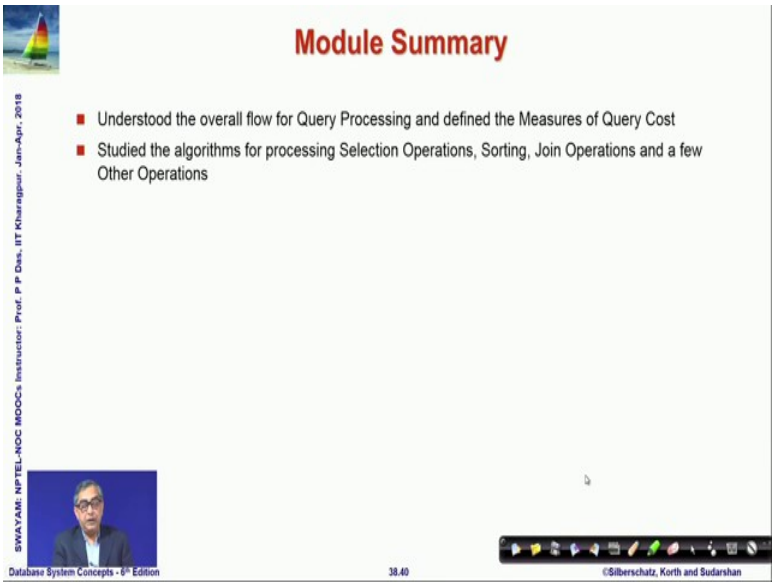
(Refer Slide Time: 35:37)



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)



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

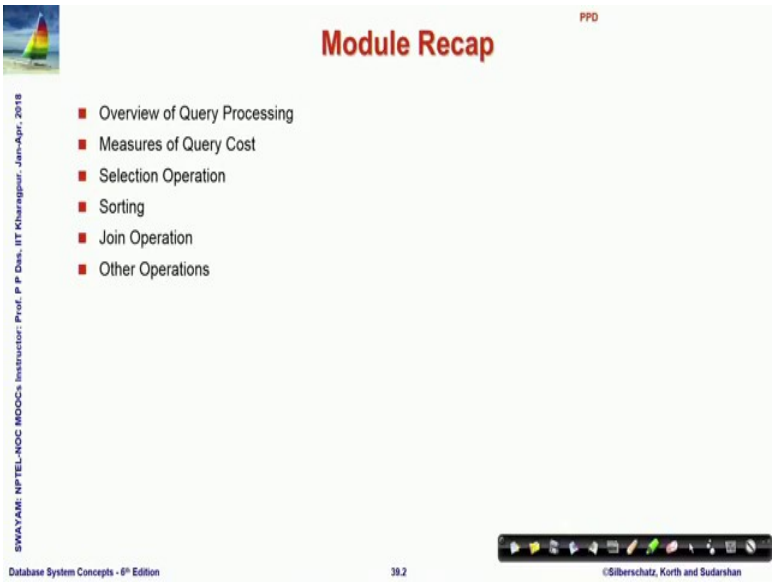
In the next module we will talk about elementary optimization strategies for processing of queries.

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

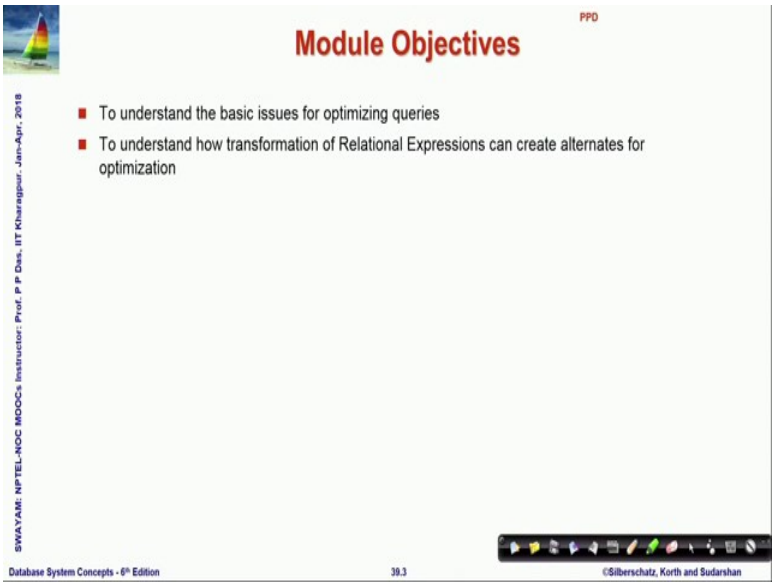
Welcome to module 39 of database management systems, we have been discussing about query processing and optimization, in the last module.

(Refer Slide Time: 00:25)



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.

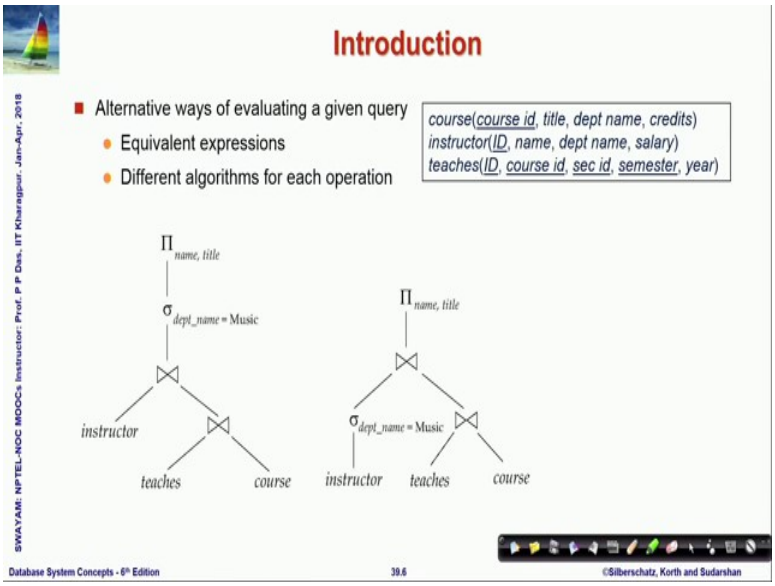
(Refer Slide Time: 01:10)



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)



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⋈teaches⋈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.

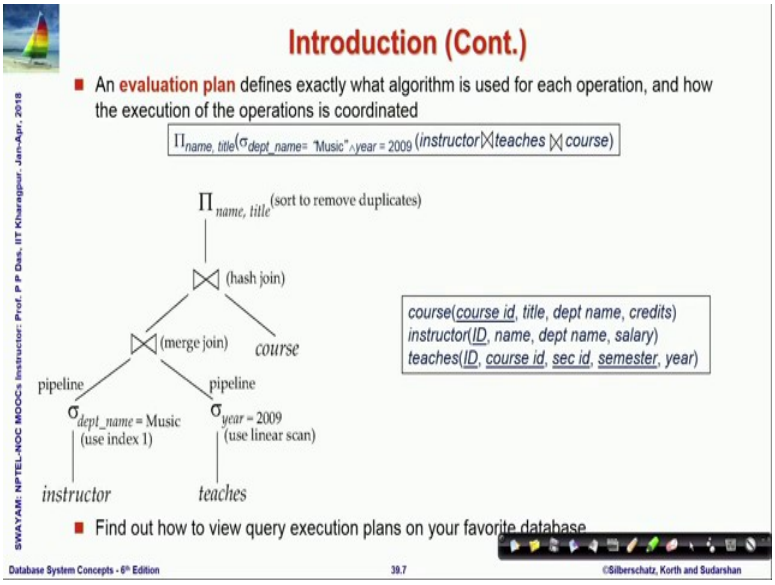
So, if we write the query in equivalent relational form this is what we will get to see and this is basically is a first join happening here then the next join happening, find next the σ and finally, the $\Pi_{name,title}$ happening here which will give us the result of this query. Now what we observe 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 $\sigma_{\text{dept\_name= music}}$ 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 $\sigma_{\text{year=2009}}$, which happens only in the teacher's relation.

So, instead of actually filtering it after the join $(\sigma_{\text{dept\_name= music}} \bowtie \sigma_{\text{year=2009}}) \bowtie \textbf{course}$ do $\Pi_{\text{name,title}}$ ($(\sigma_{\text{dept\_name= music}} \bowtie \sigma_{\text{year=2009}}) \bowtie \textbf{course}$) . 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.

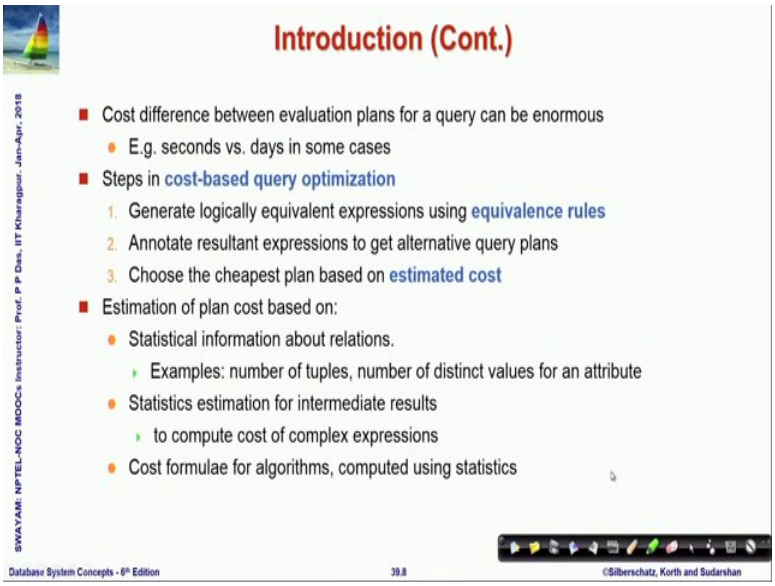
So, if we want to find out the tuples where the year is 2009 and there is no specific index on that or anything to for doing this selection as we have seen earlier the best way would

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)



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.

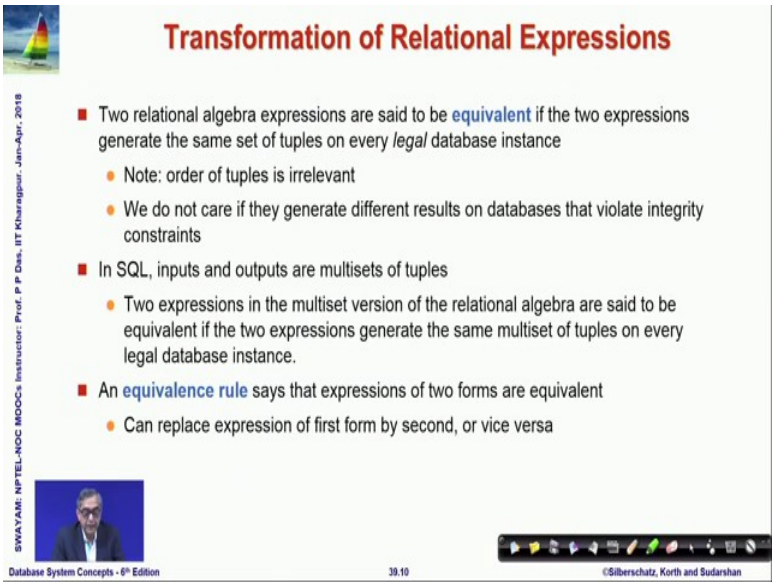
So, typical steps in this kind of query based optimization would be to first generate the candidates I am sorry first generate the candidates that is generate the equivalent expressions that is given a query I have one relational expression and we would like to generate equivalent expression using a 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.

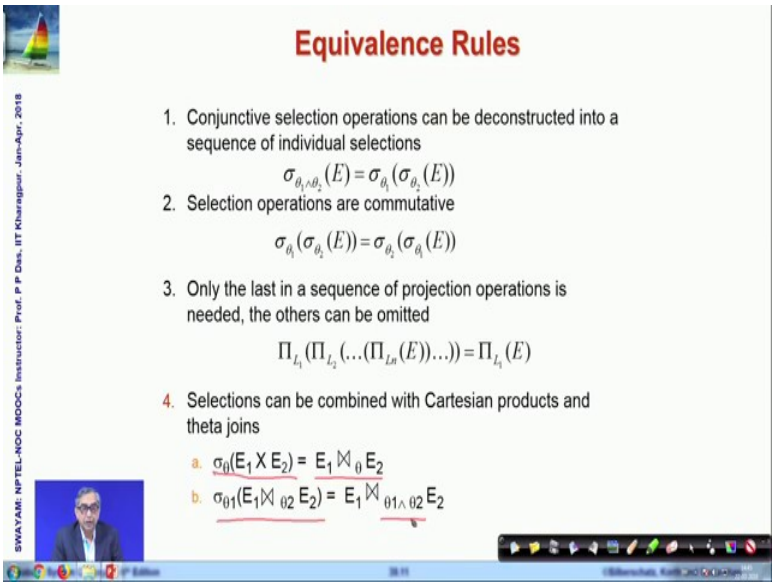
(Refer Slide Time: 10:27)



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)



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, $\sigma_{\Theta 2}$ 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.

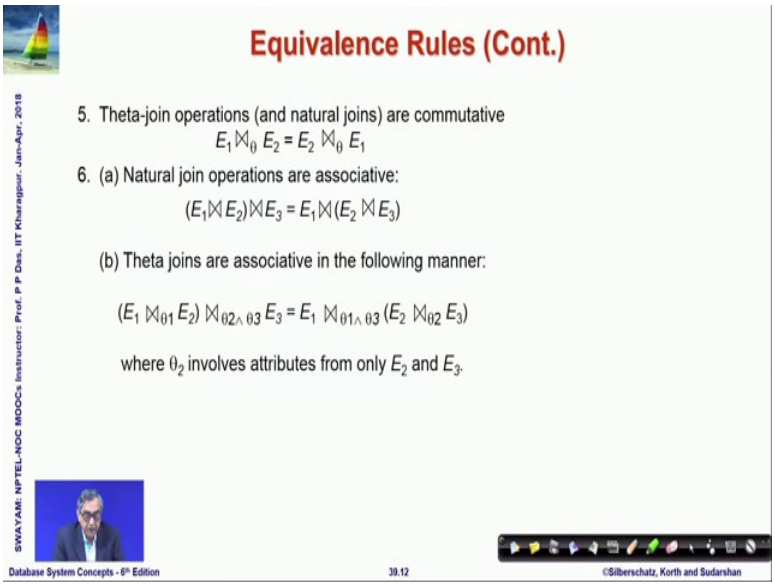
So, applying these two rules we get three relational expressions which are all equivalent this, this and this are all equivalent, but if you actually think in terms of processing the query and the cost involved you will be able to figure out that naturally the cost of doing this may be relatively more 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)

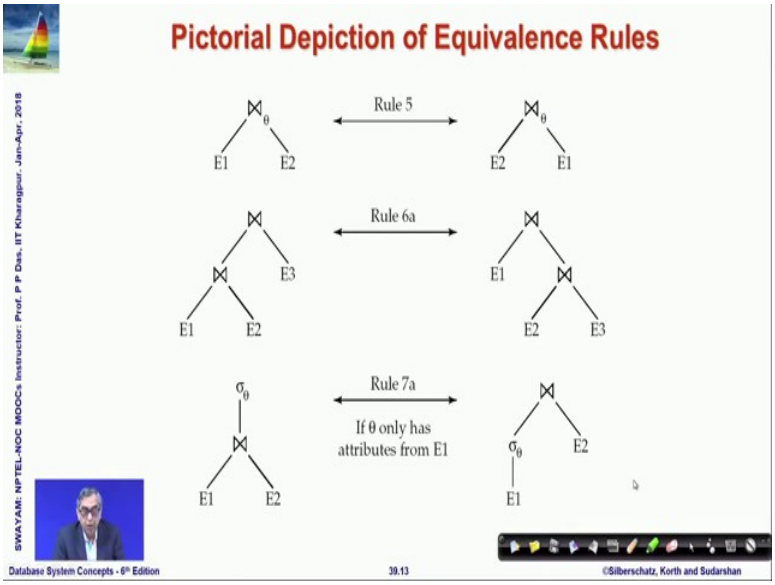


Then Θ join operation are natural or natural join operations are commutative, we can change interchange their relations.

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 $(\mathbf{E_1} \bowtie_{\Theta 1} \mathbf{E_2}) \bowtie_{\Theta 2 \wedge \Theta 3} \mathbf{E_3}$ 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, $\mathbf{E_1} \bowtie_{\Theta 1 \wedge \Theta 3} \mathbf{(E2} \bowtie_{\Theta 2} \mathbf{E3)}$ 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.

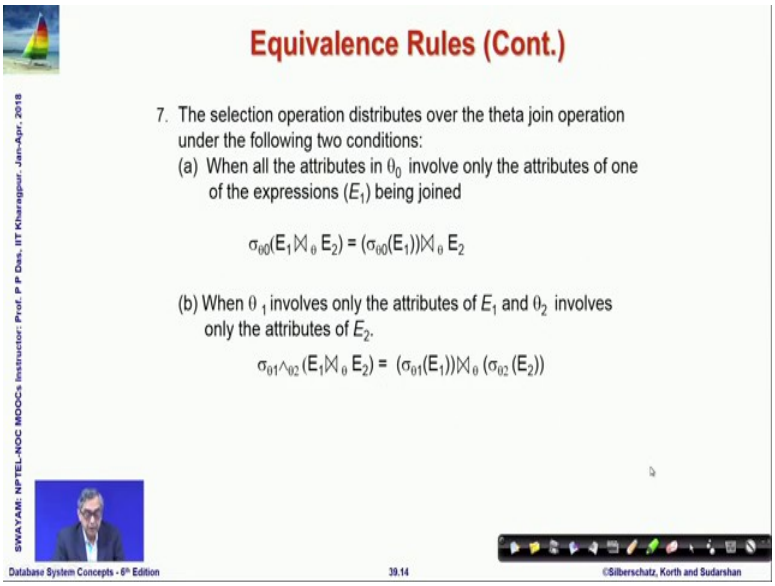
(Refer Slide Time: 16:01)



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, $\mathbf{(E_1} \bowtie \mathbf{E_2)} \bowtie \mathbf{E_3}$ here you are saying first you do E_2, E_3 and then you join with E_1. $\mathbf{(E_2} \bowtie \mathbf{E_3)} \bowtie \mathbf{E_1}$ 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 contents 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.

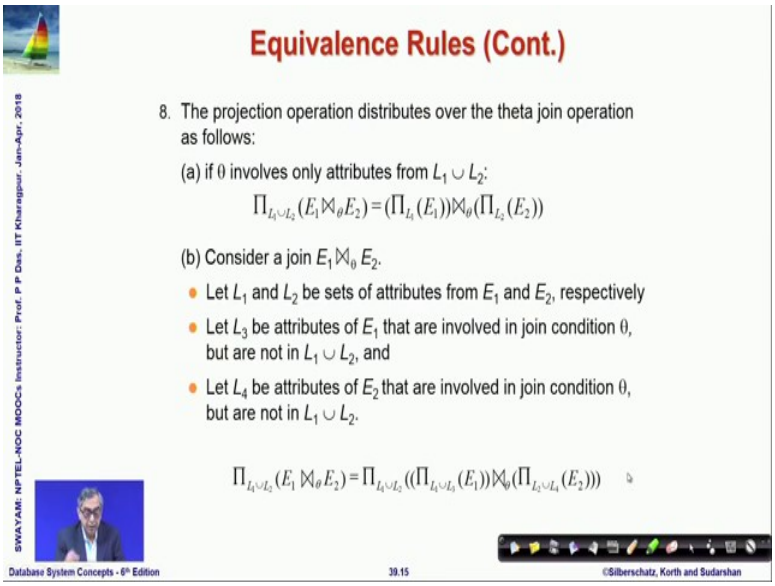
(Refer Slide Time: 17:00)



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 $\boldsymbol{\sigma_{\Theta 1 \wedge \Theta 2}(E_1 \bowtie_\Theta E_2)}$ 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 E2, $\boldsymbol{\sigma_{\Theta 1}(E_1) \bowtie_\Theta \sigma_{\Theta 2}(E_2)}$. These are these are pretty straightforward rules if you think about the corresponding set theoretic reason.

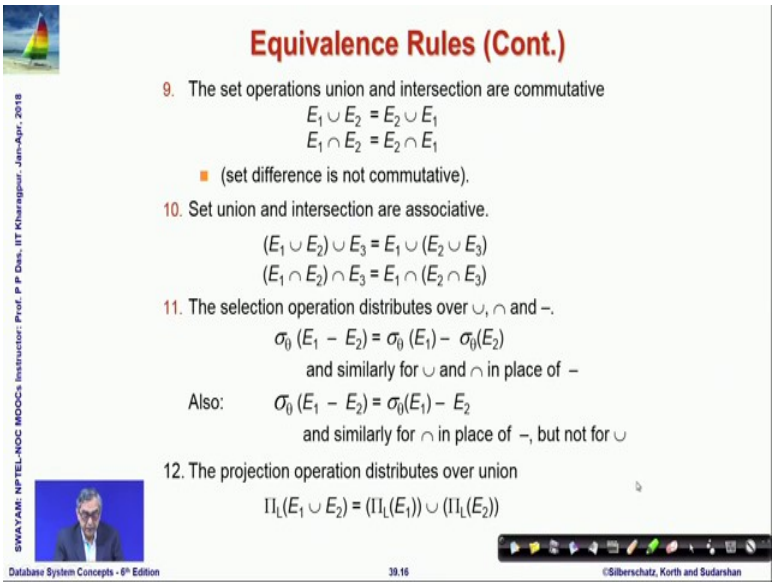
(Refer Slide Time: 17:58)



So, the other rules are 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.

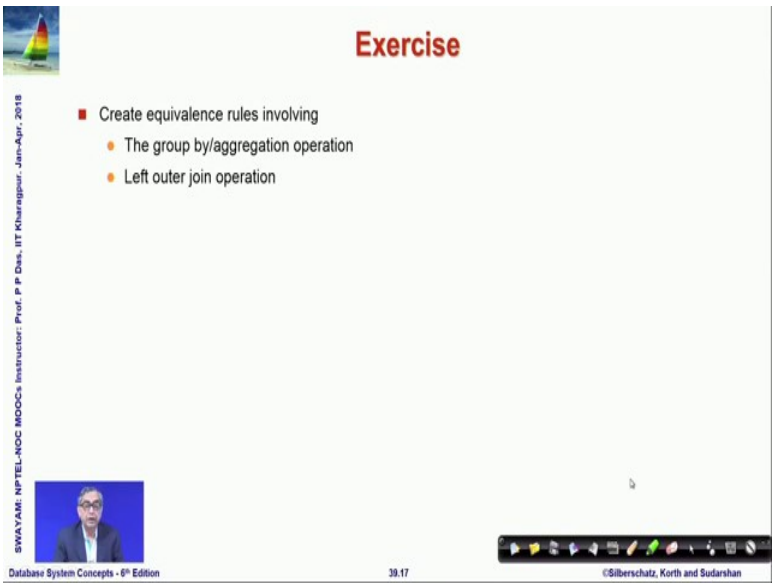
So, if L_3 be the attributes of E_1 that are involved in the join condition Θ and L_4 are what are. So, you have the join condition Θ. So, what you are saying that the attributes L_3 of you are not involved here and attributes L_4 of E_2 are involved here, but they are not so, in terms of **L1 U L2** 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)



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.

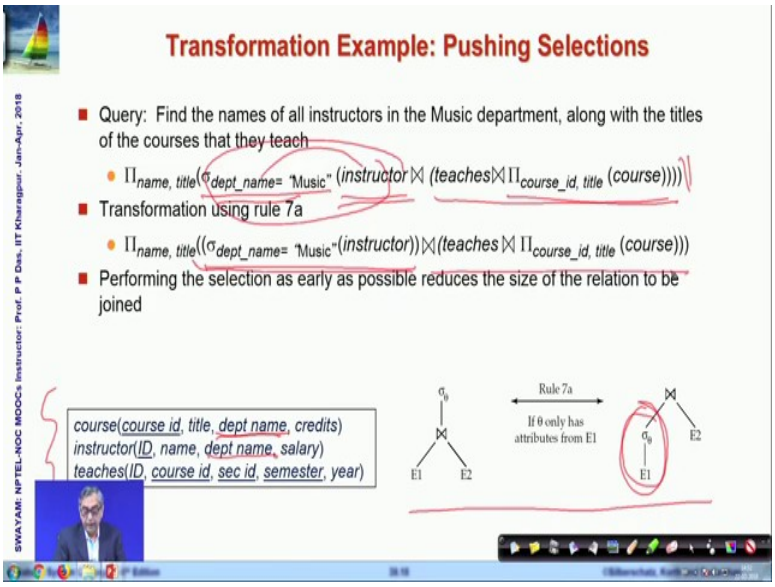
(Refer Slide Time: 19:35)



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)



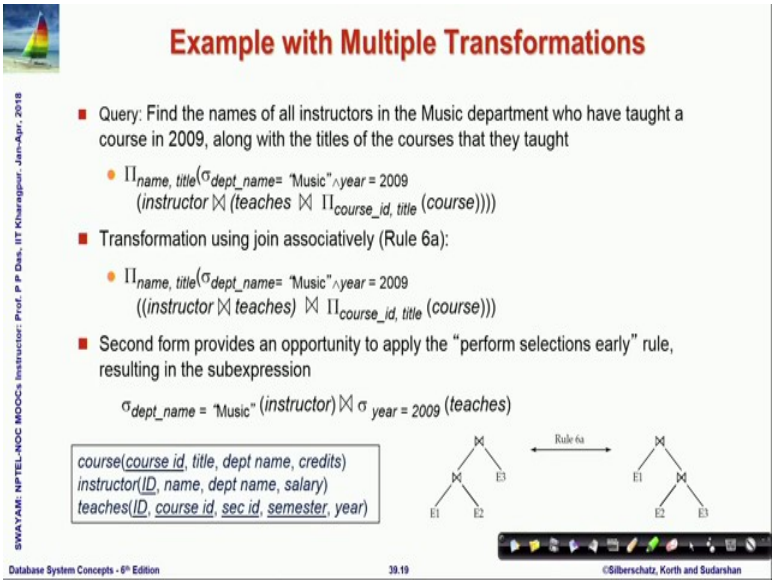
Let us move on to a couple of examples so, here is a query here the relations from the university database, here is a query find the names of all instructors in the music department along with the titles of the course they teach. So, a while ago we saw this so, this is what the query is in the relational 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)



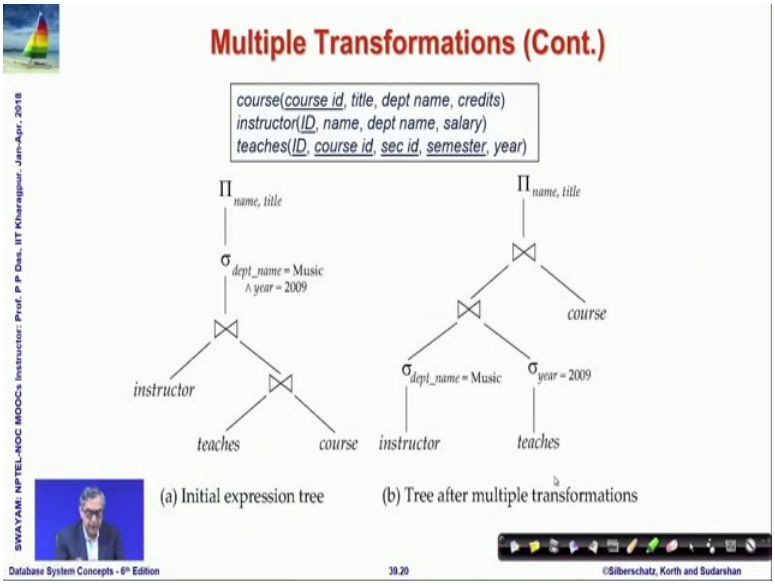
So, this is one example, this is another example query we are taking find the names of all instructors in the music department, we have taught a course in 2009 along with the titles of the course they taught. So, here is a the corresponding here is the corresponding relational query which you can convince 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.

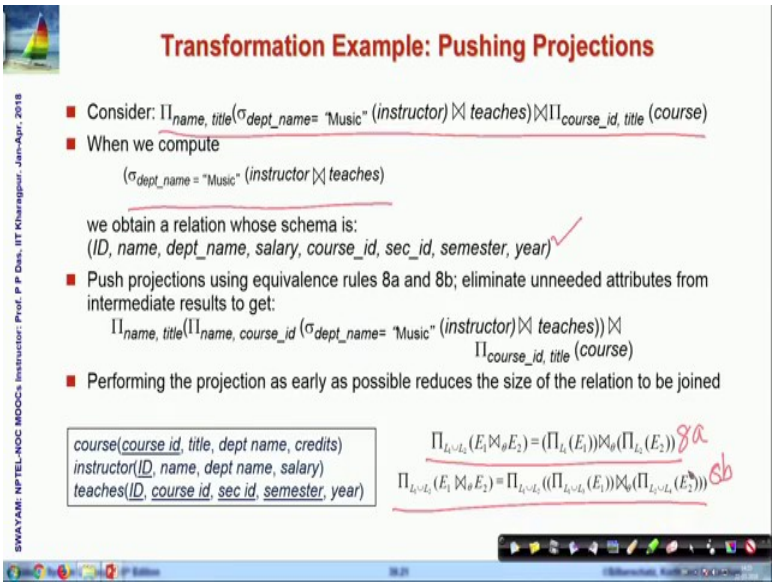
So, naturally each one of these will become much smaller corresponding to the whole instructor or teaches relation therefore, their join will also be smaller. So, which means eventually this part this part of the query will become much smaller in size in the result and the consequent second join would 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 a 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.

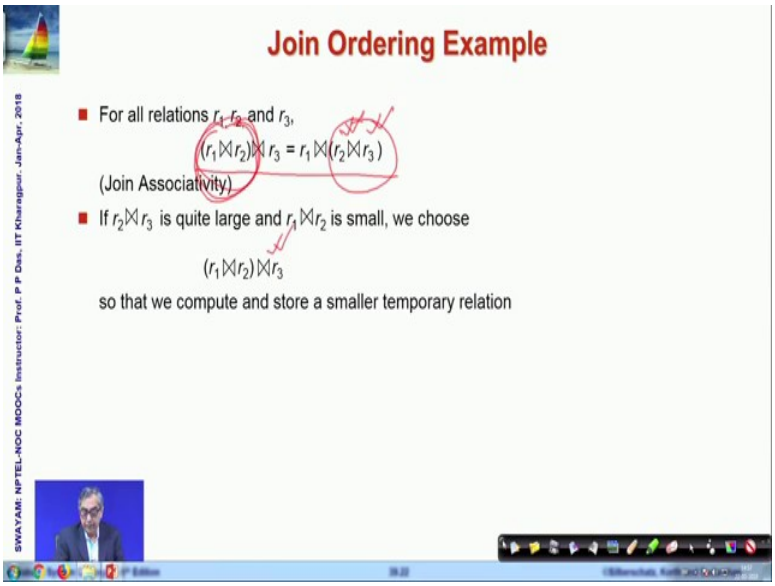
(Refer Slide Time: 24:12)



And then certainly they give you a significant advantage and now this is an example which show that you can push the projection.

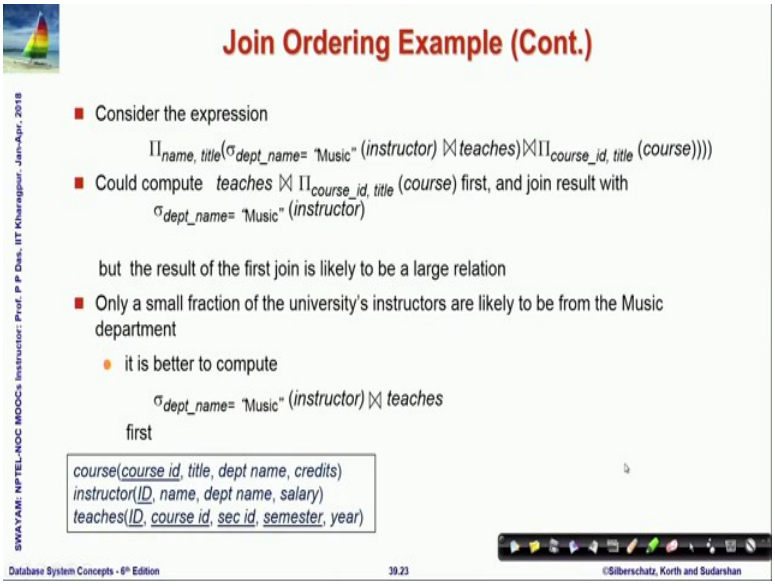
So, here is what you wanted to do and while we compute this we will get a relation of this form and we can push, we can use these rules rule 8a and 8b, this is rule 8a, this is rule 8b by this we can push the projections inside and make the relations smaller because now if you push the projection then 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)



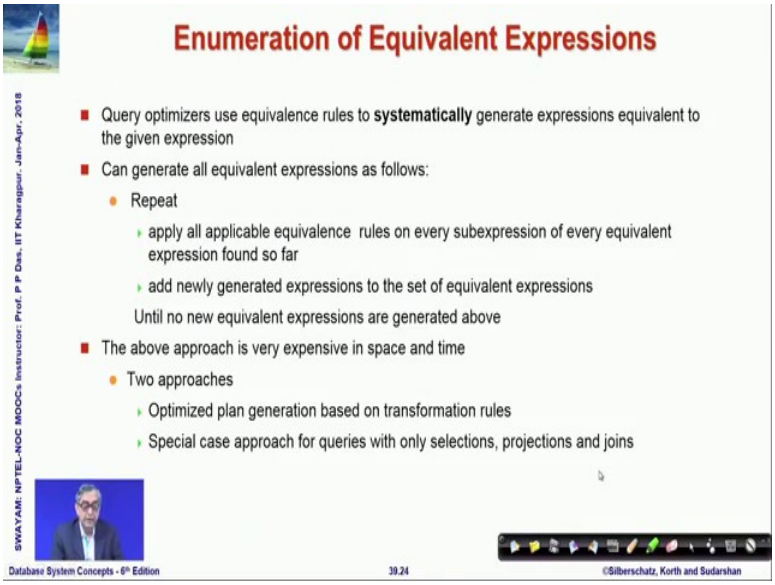
You can also the these are examples where you can take care of the fact that you can reorder joining to get better result for example, the you can if I have to do join three relations like this, I could do either this first or this first. Now if I do this first ($\mathbf{r_1} \bowtie (\mathbf{r_2} \bowtie \mathbf{r_3})$) 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 r3 (($\mathbf{r_1} \bowtie \mathbf{r_2}$) $\bowtie \mathbf{r_3}$)). 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)



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.

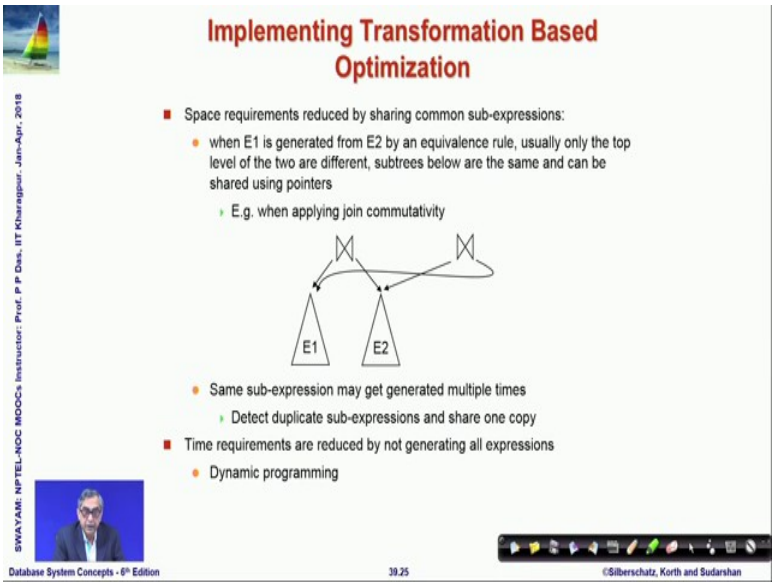
(Refer Slide Time: 26:20)



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)



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.

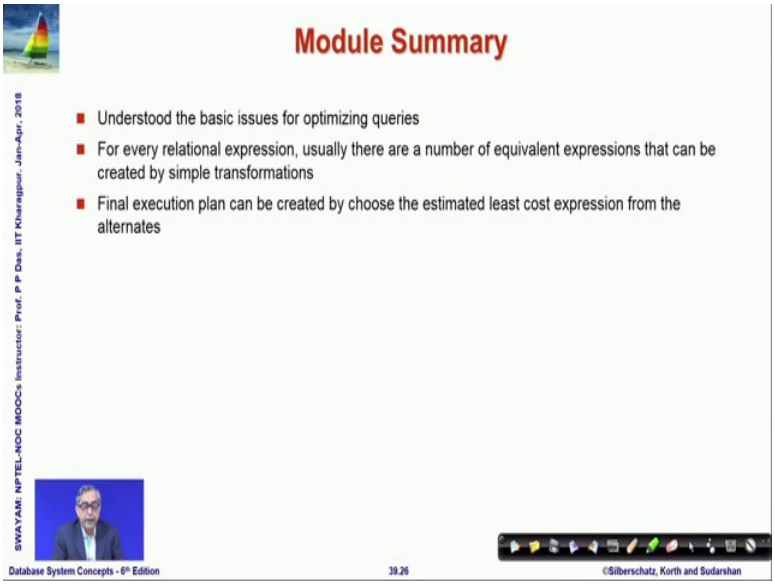
So, if that be the case then in terms of generating the alternate we do not really need to keep or evaluate all of the expressions the whole of the expression in every alternate. We could actually do something like sorry we could actually do something like we could have a join and then instead of really 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)



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.

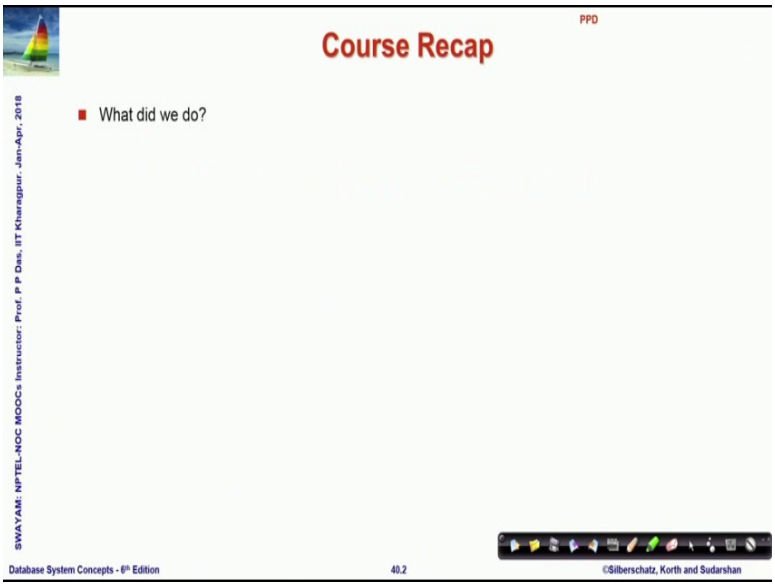
And finally, an evaluation plan is made where specific choice is made for the different algorithms for doing different operations and that final plan which is which has the best cost is passed on to the query processing engine to query evaluation engine. And that then goes forward and does the B tree 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

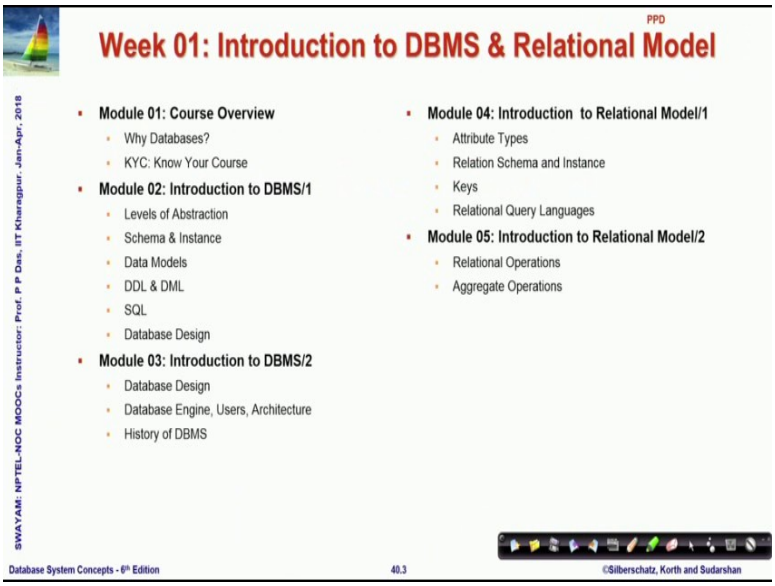
Welcome to module 40 of Database Management Systems. This is for course summarization this is the last module of the course.

(Refer Slide Time: 00:23)



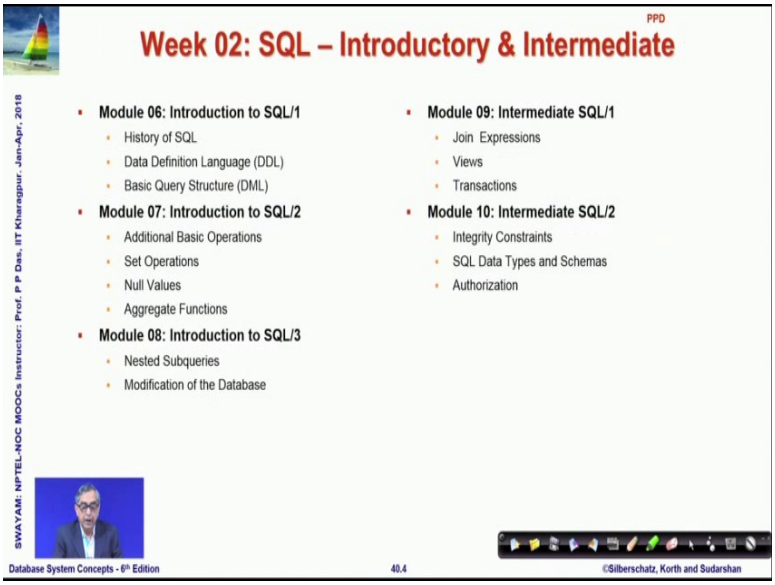
So, I would just start with doing a quick recap of what all did we cover and what we expectedly learnt.

(Refer Slide Time: 00:32)



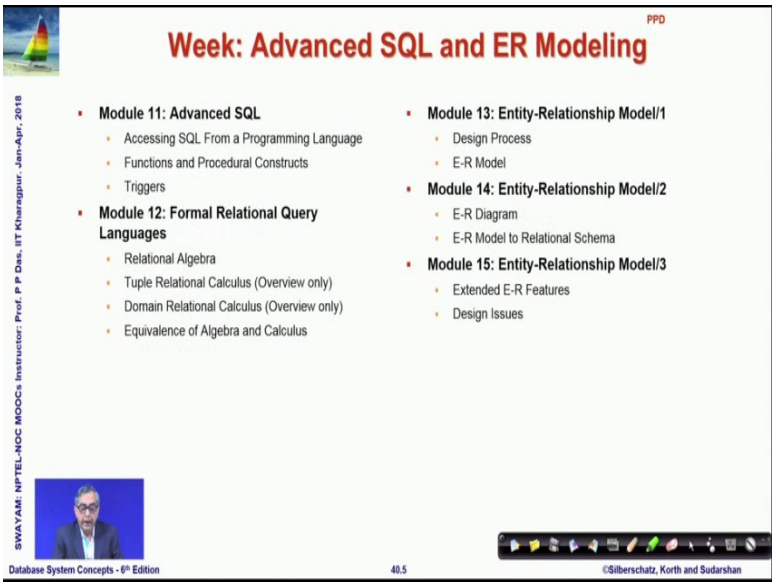
In the week 1, we talked primarily of Introduction to Database Management System and the Relational Model which is the foundation of a database system.

(Refer Slide Time: 00:42)



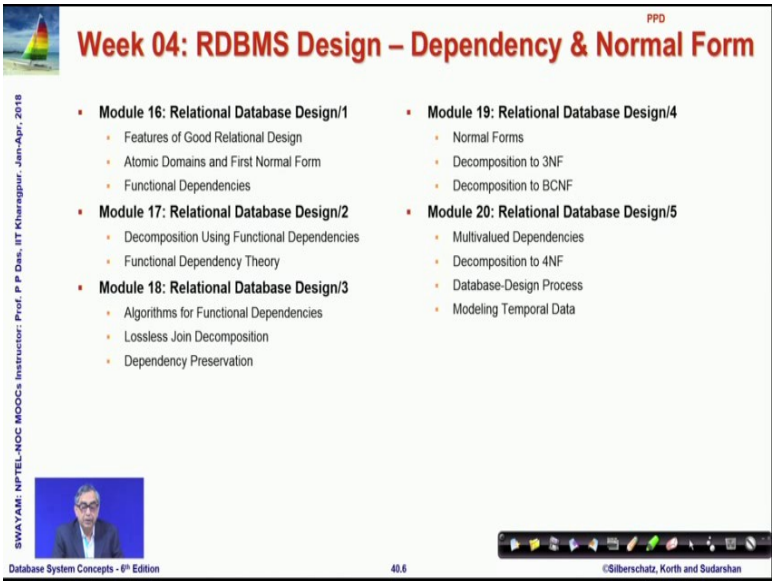
In week 2, we started off with query language SQL at an Introductory level and then at an Intermediate level which are really really the first major aspect of a database particularly relational database that a student must master.

(Refer Slide Time: 01:01)



In week 3, we continued with the advanced SQL and did the aspects of modeling from specification in terms of Entity Relationship Model.

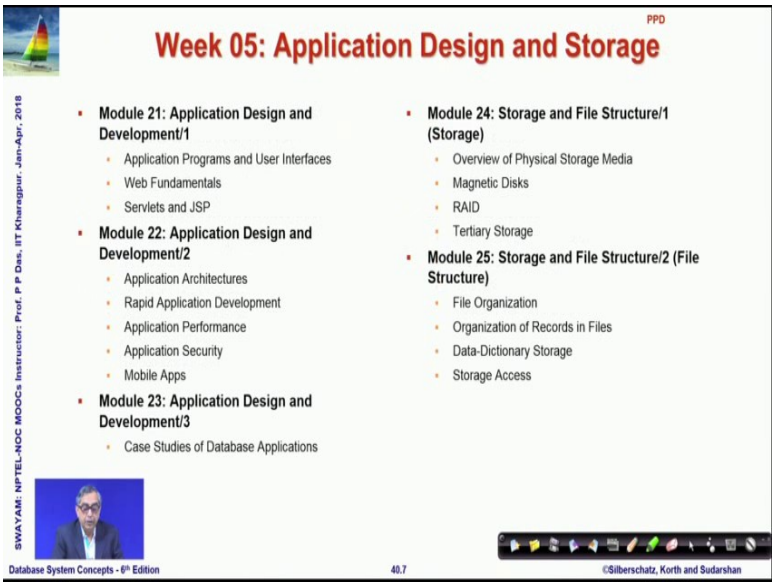
(Refer Slide Time: 01:12)



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.

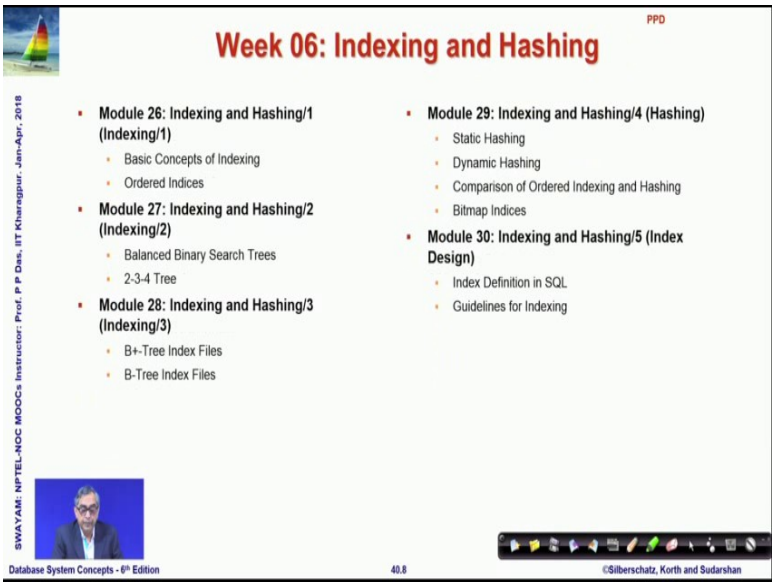
So, this is based on dependency and different normal forms and I am sure you have spent a good time on mastering these.

(Refer Slide Time: 01:40)



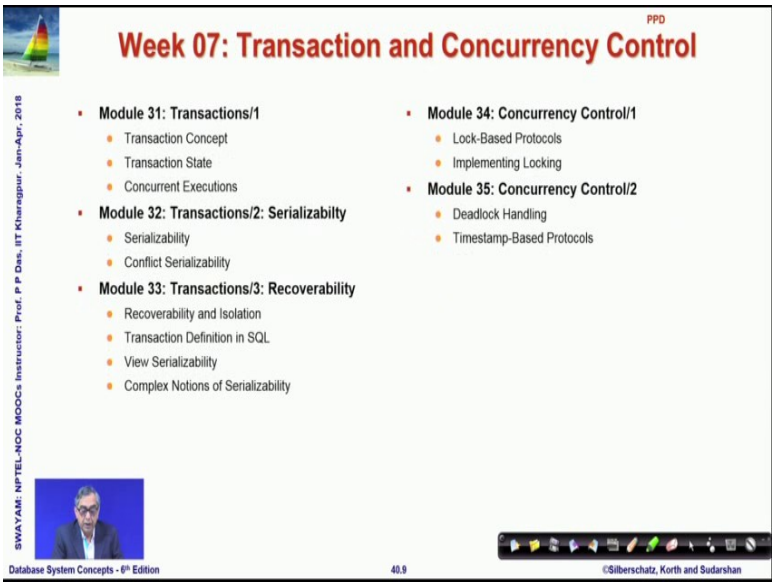
We followed up in week 5 with application design and discussing aspects of storage structure, how will actually the items, data items be stored in the different memory and disk structure.

(Refer Slide Time: 01:56)



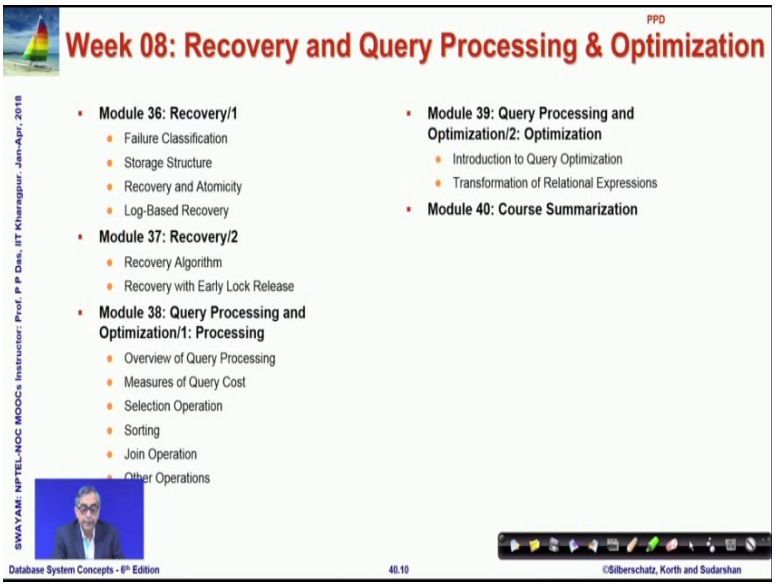
In the following week, in week 6, we discussed about indexing and hashing to for making the accesses really efficient.

(Refer Slide Time: 02:05)



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.

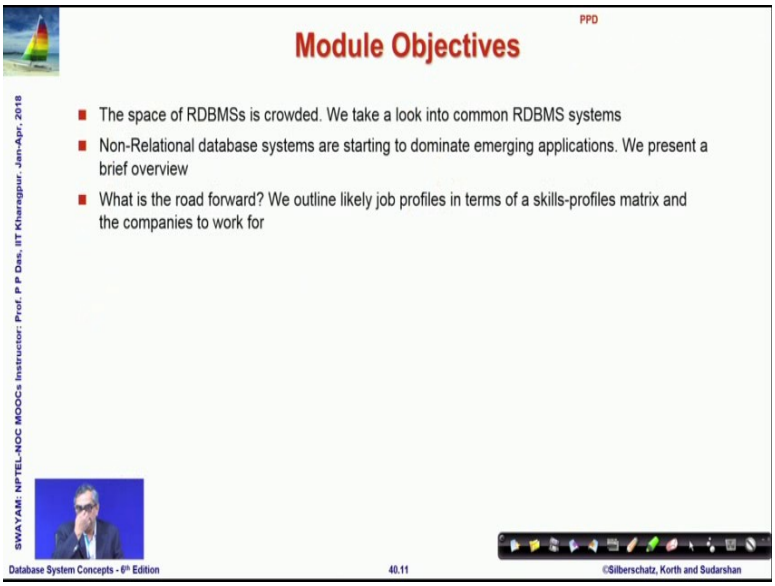
(Refer Slide Time: 03:03)



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.

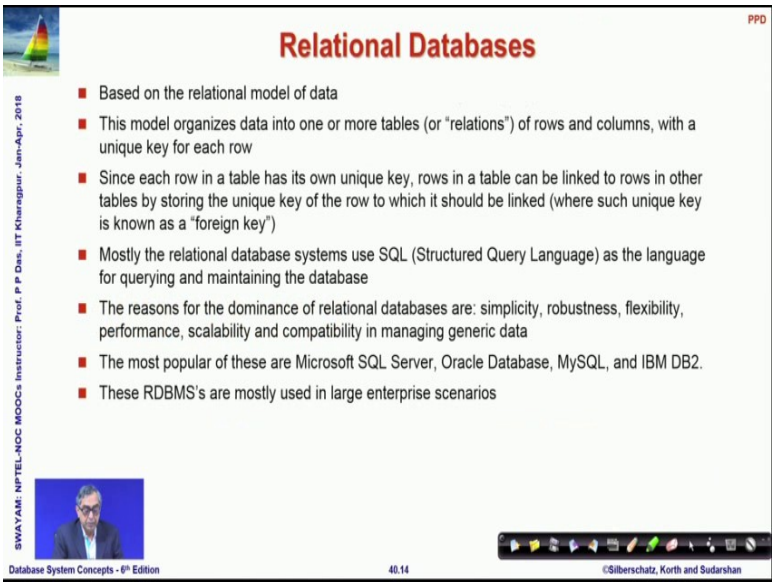
(Refer Slide Time: 03:48)



So, given that 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 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.

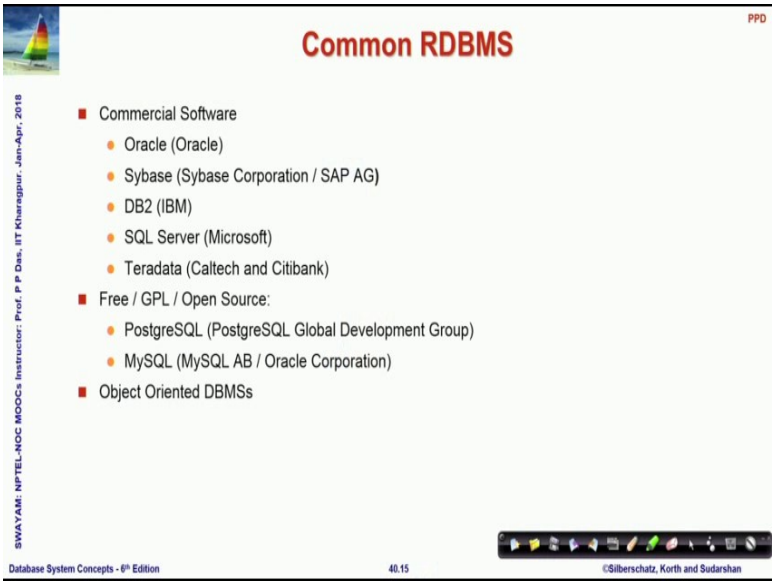
And then finally, I would like to conclude with what should be the road forward from you, presenting you a kind of a skill profile matrix so that what skills you must pick up to actually get a job off certain profile and what are the companies that you might look at working for.

(Refer Slide Time: 04:55)



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)



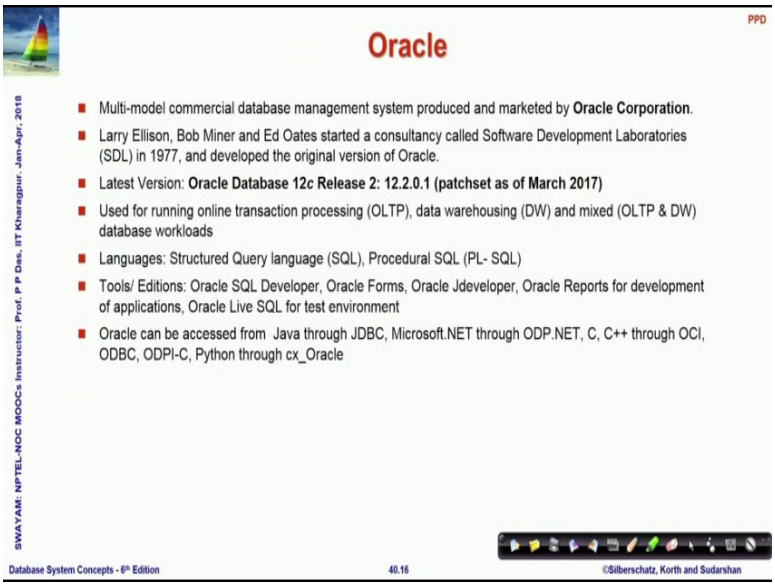
Now, these are the common database systems. So, I have chosen the ones which are most widely used, most easily accessible and kind of large companies use them, large databases exist on them. So, there are primarily 2 classifications; one is a set of database

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, I 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 my SQL which is was originally from a Swedish company called my SQL 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.

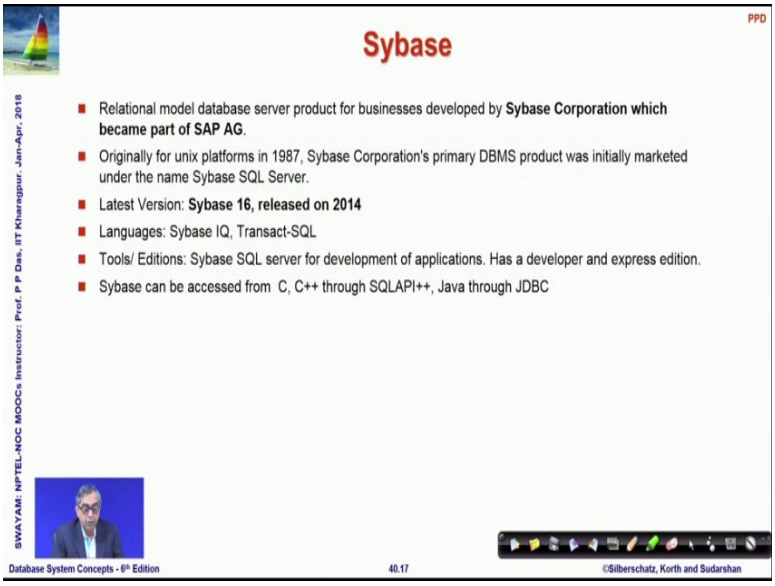
So, if you look in through these, then in most cases you will find in terms of the gross functionality of the kind of SQL that you can write, a large subset of the SQL that you can write on databases maintained through these database systems will be same. So, what you have learnt here would be applicable 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)



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.

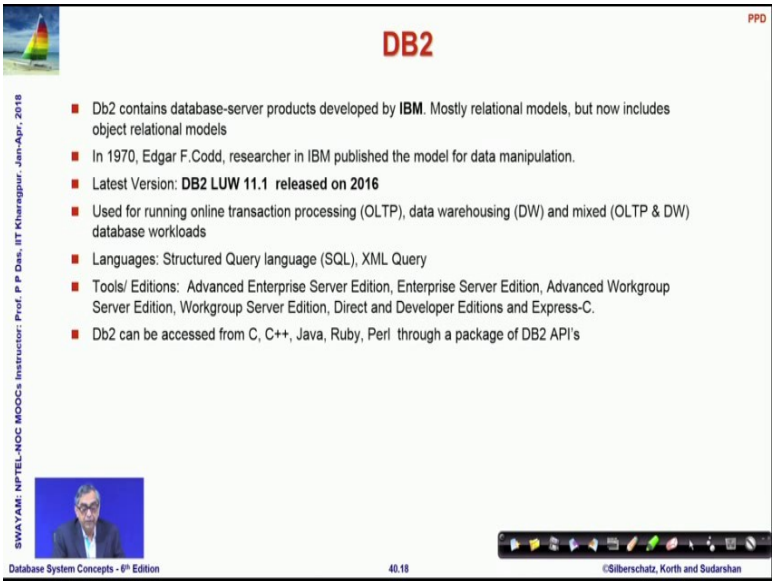
(Refer Slide Time: 08:13)



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

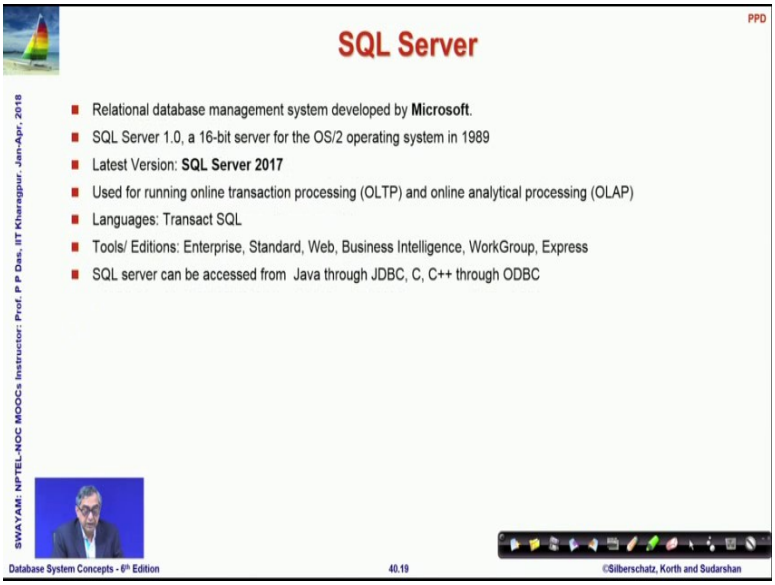
half or 4 years ago and, but Sybase is a has been a very good database systems for programming through API's and it has really good support for that.

(Refer Slide Time: 08:42)



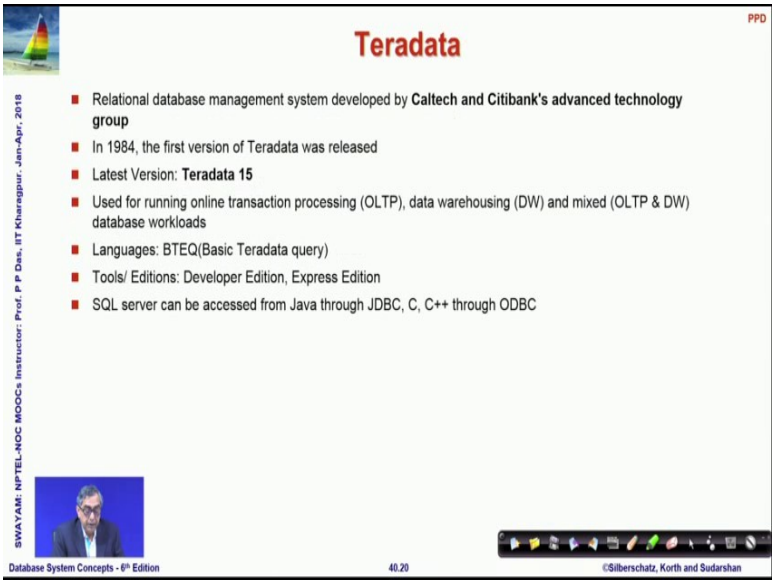
DB 2 is also a very old possibly the oldest surviving database systems which started in 1970. So, when almost the E. F Codd of the Boyce Codd normal form published the data manipulation schemes from IBM. So, this is also a widely used, last release 2016.

(Refer Slide Time: 09:07)



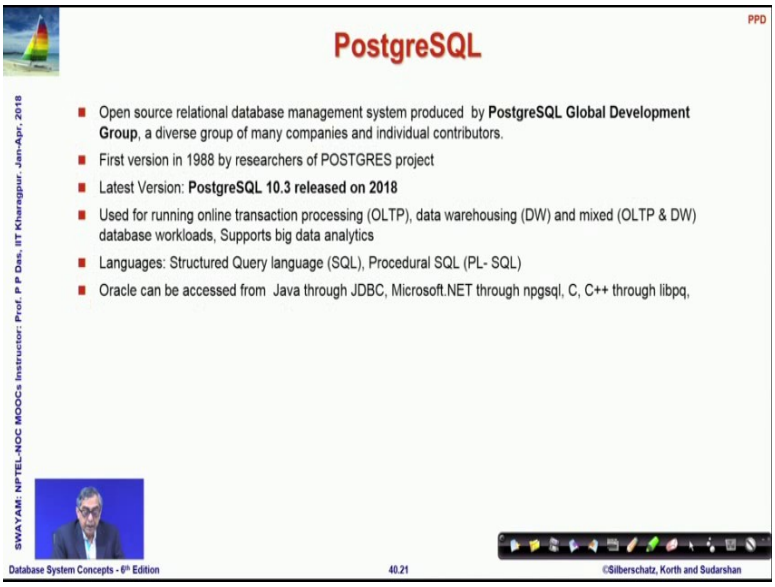
Microsoft the started database systems in 1989, last release happened last year and that is very widely used if you are particularly on windows system, it is one of the most popular one in terms of the windows operating system.

(Refer Slide Time: 09:24)



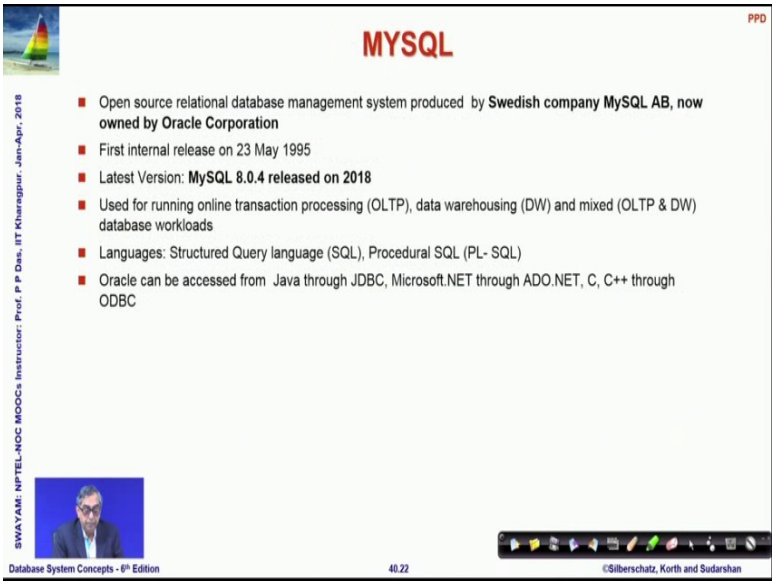
Teradata is relatively new. It was released in 1984 and it is, but it is it is one where lot of new developments are still happening and new experiments keep on happening and the current version is a Teradata 15.

(Refer Slide Time: 09:43)



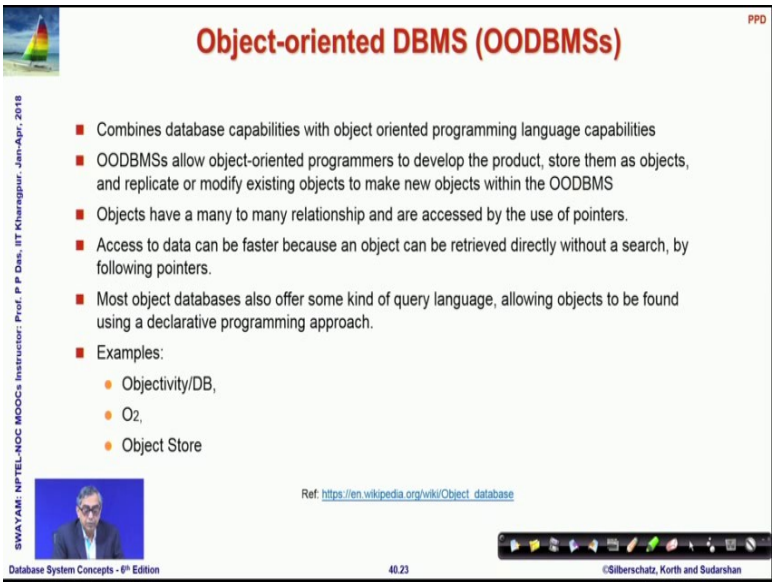
And in terms of the ma free or open source GPL databases Postgres started in 1988 about 30 years back and as I said, this is this is has a release even half of this year.

(Refer Slide Time: 10:04)



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.

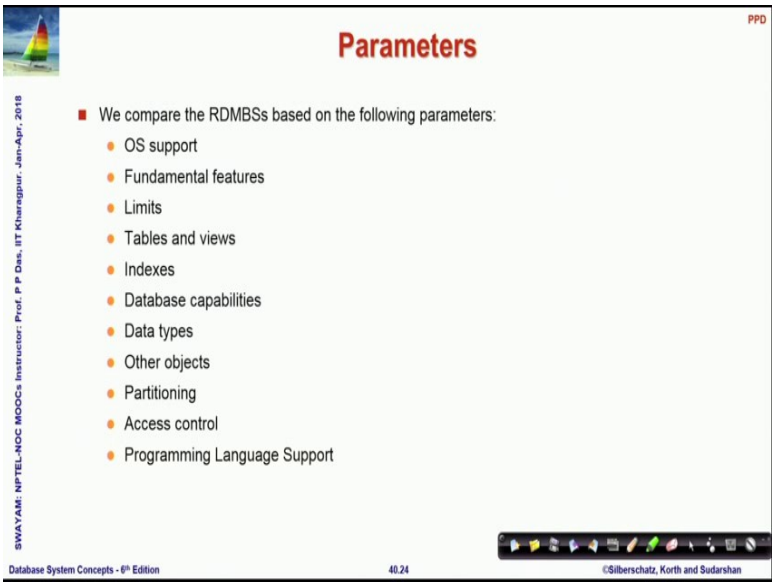
(Refer Slide Time: 10:34)



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)



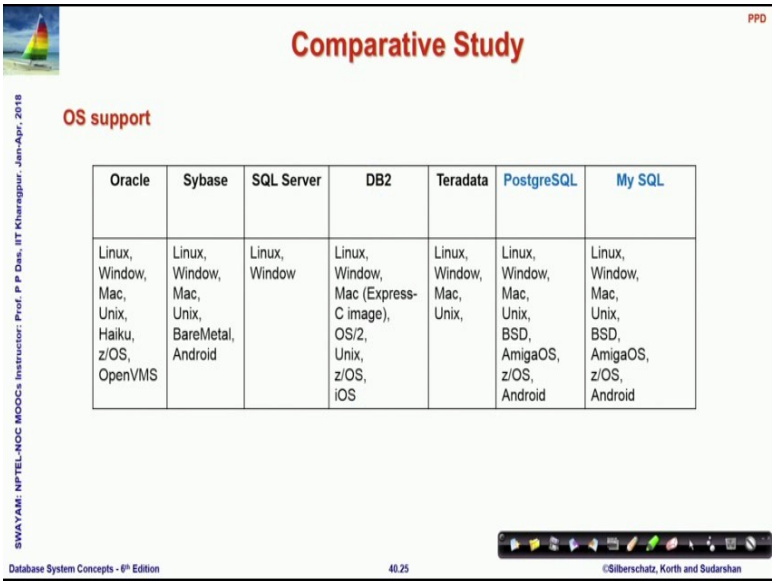
Now, when you come across a particular system that your company or your university needs to use and you would like to choose, then it will be good to look at the different aspects of that system. These are here are some of the parameters on which these database systems vary in a in a minimal to a very 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)

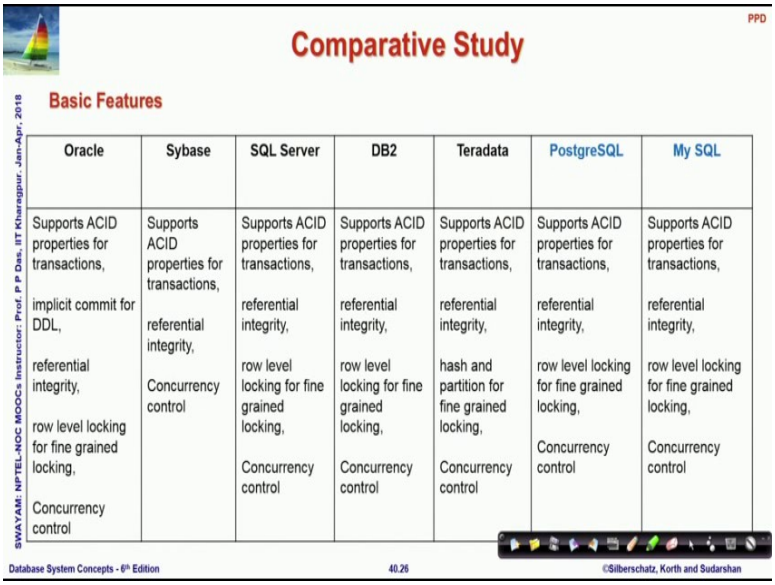


But here after given a compilation of different you know on different aspects, how do these common database RDMS systems agree or differ. So, this is like a slide which shows what are the operating system support for different databases. So, if you are for example, working on android, then you can easily 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

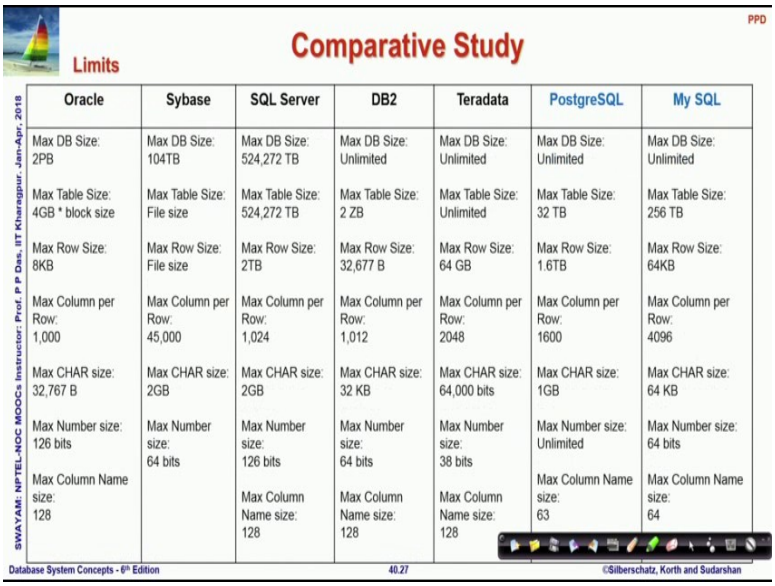
medium scale or low cost applications the free database systems, Postgres and MySQL are really good options there are different such features.

(Refer Slide Time: 15:23)



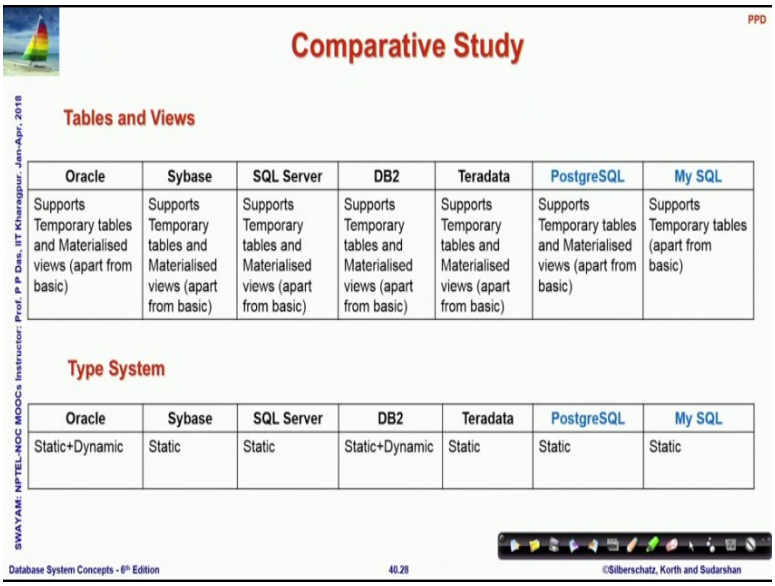
The basic features are compared to here.

(Refer Slide Time: 15:27)



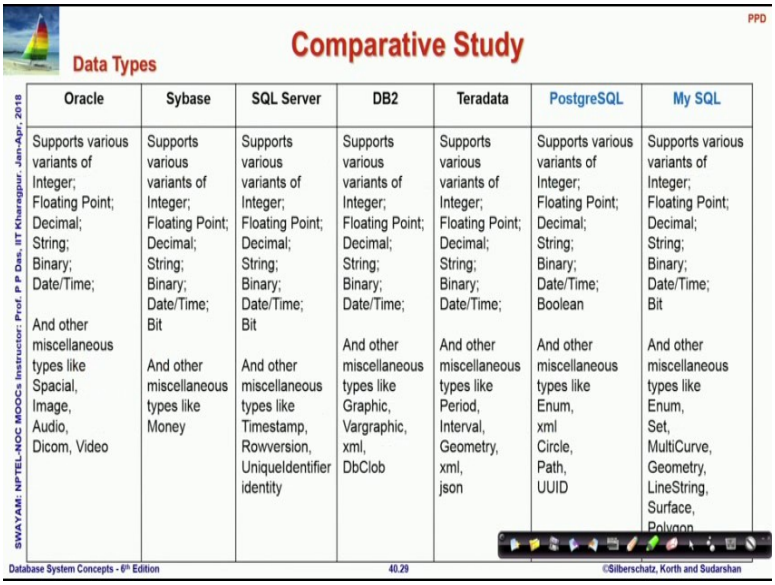
At compile different limits of different database systems here. So, you can see in terms of maximum row size, columns per row and so on and so forth, how do they differ. So, if you are making a choice in terms of what database I will use.

(Refer Slide Time: 15:42)



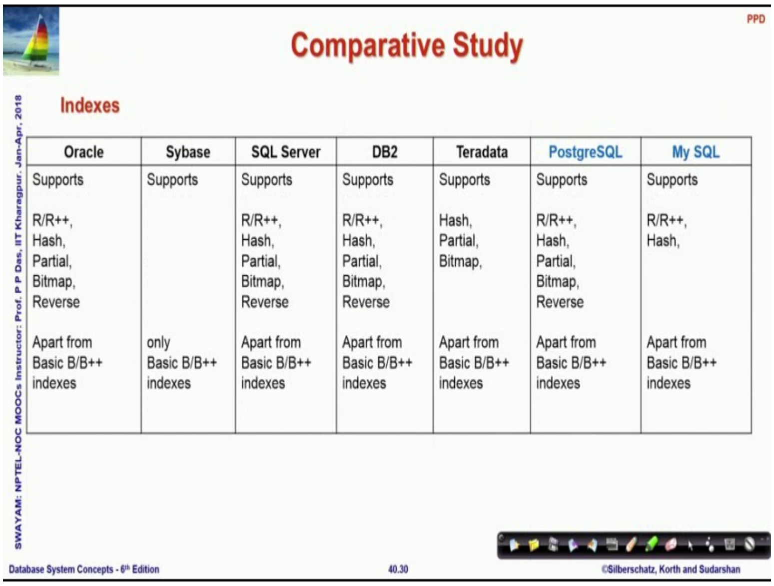
You can use these information. This talks about tables we use the type systems, what kind of typing is used.

(Refer Slide Time: 15:48)



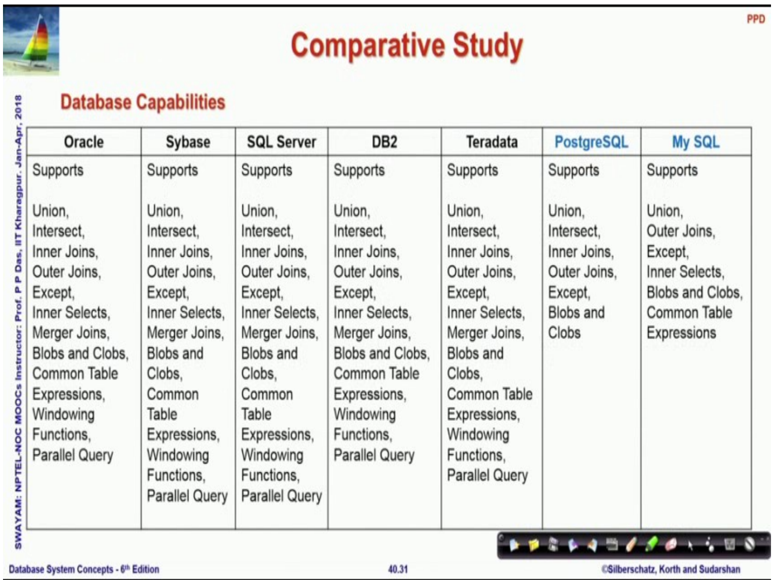
The different data types that are used are given here.

(Refer Slide Time: 15:52)



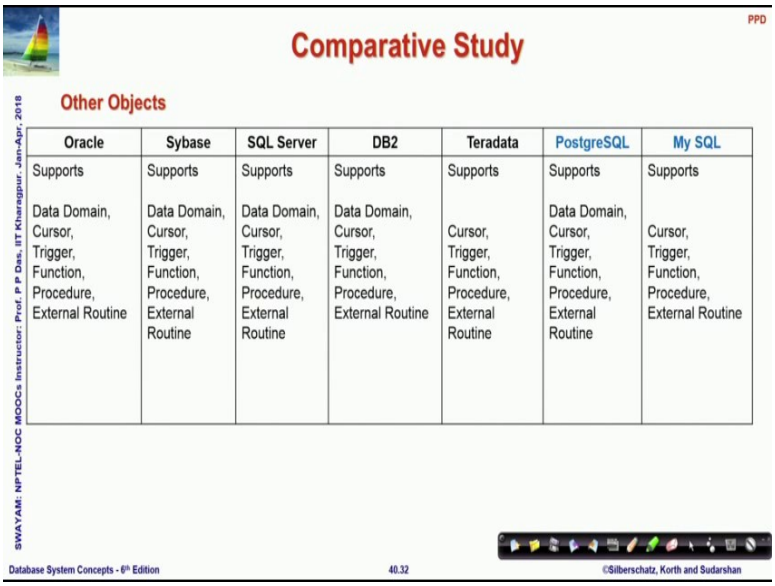
The index mechanisms are discussed here.

(Refer Slide Time: 15:57)



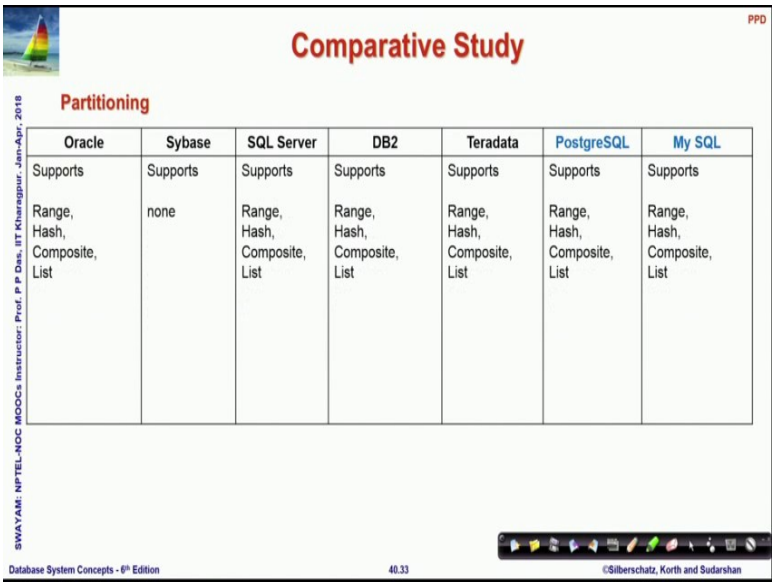
The capabilities of the database, what it can do overall.

(Refer Slide Time: 16:01)



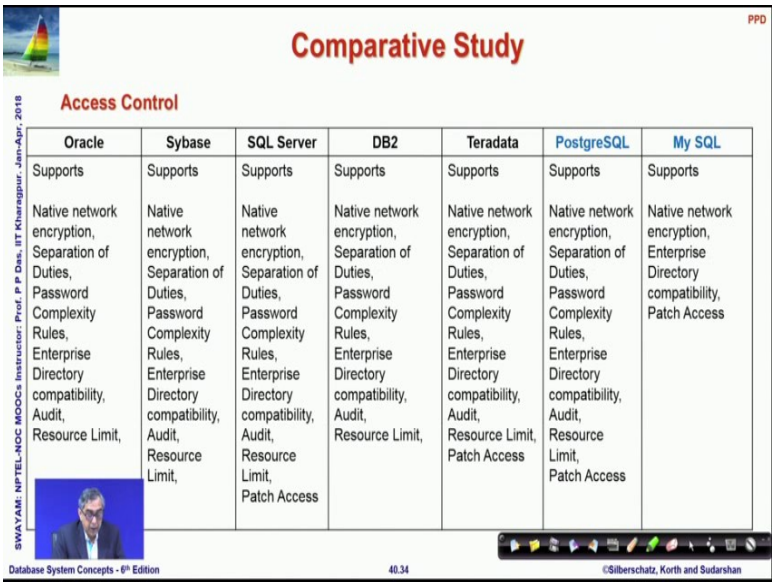
Other kinds of objects that it supports.

(Refer Slide Time: 16:05)



The different partitioning mechanism.

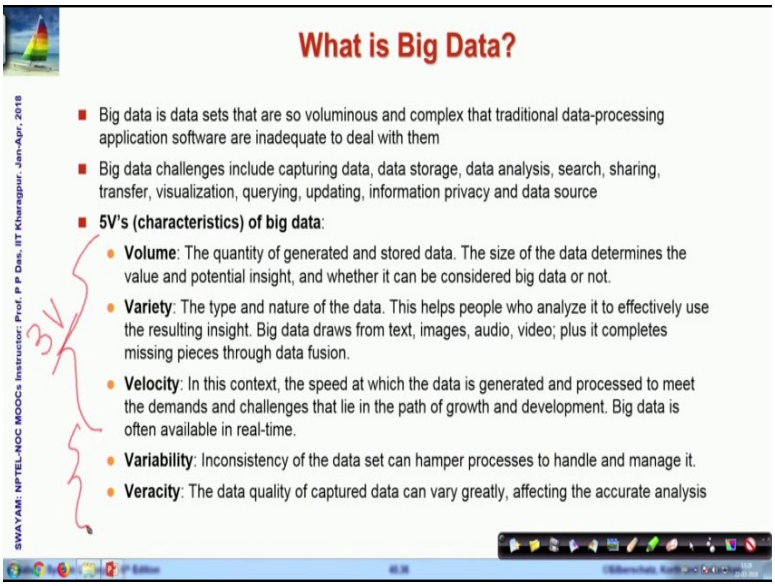
(Refer Slide Time: 16:08)



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)



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 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.

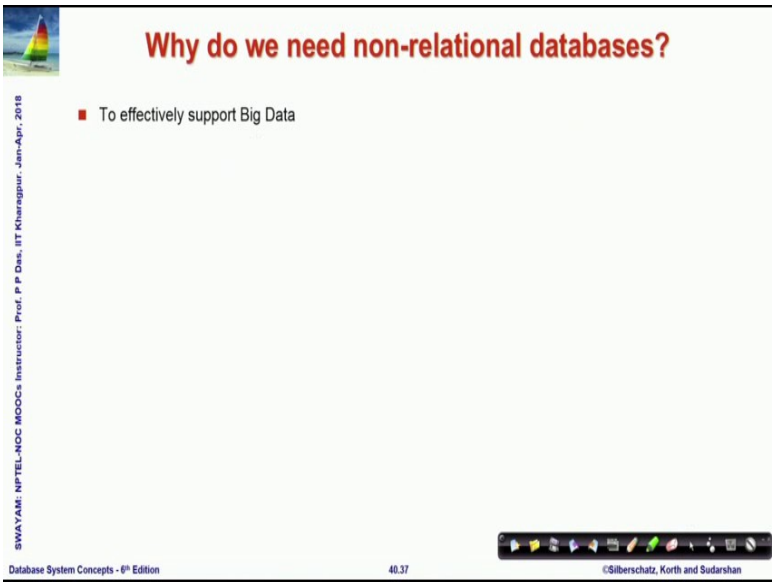
So, those Facebook comments are phrases and if I want to make certain query based on that, if I want to make a query that, how many Facebook users has commented on the success of Virat Kohli as a captain of India if I want to make such a query, then the question is how do I do that? Because it is not 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.

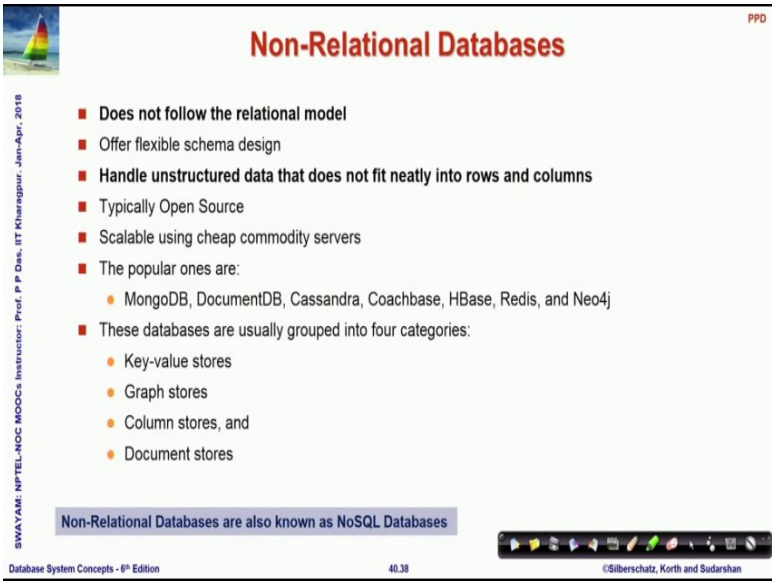
So, those are the challenges of big data, then there could be variability inconsistency of the data that you are because maintaining integrity is a big problem; there could be issues in terms of quality of the data that is called veracity. So, actually these things characteristics that I have put, there 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)



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)



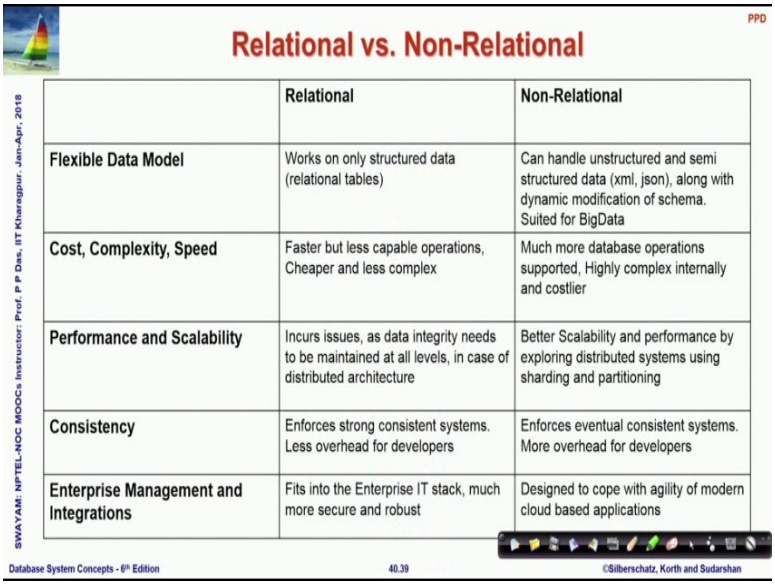
So, that in non-relational databases certainly as the name suggests, do not follow relational model, they offer flexible schema design. The schema may itself change while the database is evolving which is not the case in the relational schema is fixed, only the data can change, but here the schema itself 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)

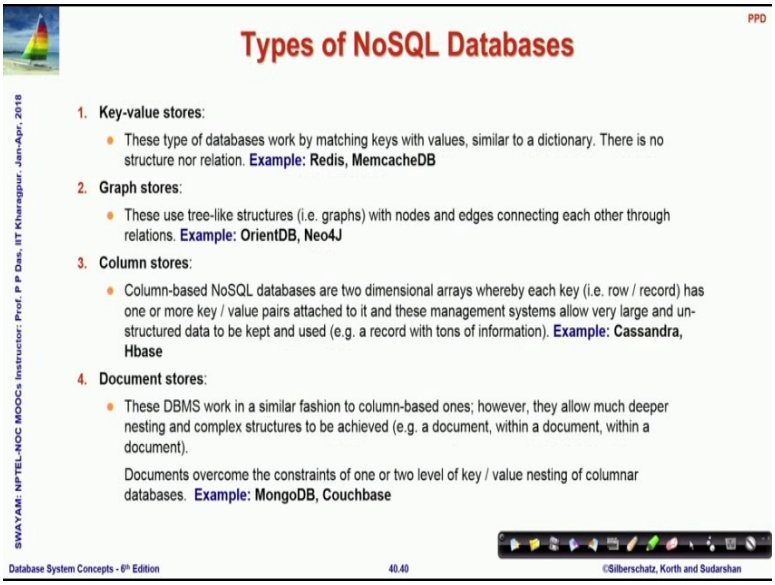


So, here is a quick comparison between the relational and the non-relational, in terms of the flexibility of the data model, relational is very structured when on relational has to have handle unstructured data, semi structured data and therefore it has to be flexible in terms of the data model, cost 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 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.

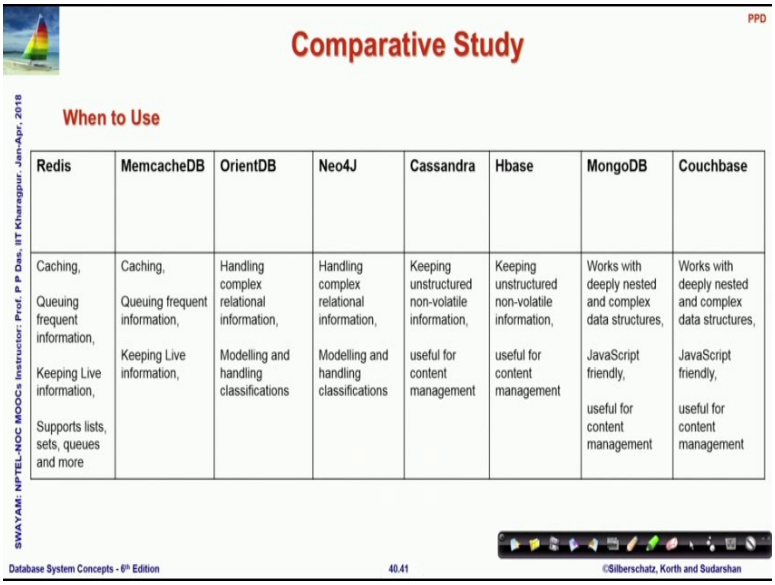
(Refer Slide Time: 25:27)



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.

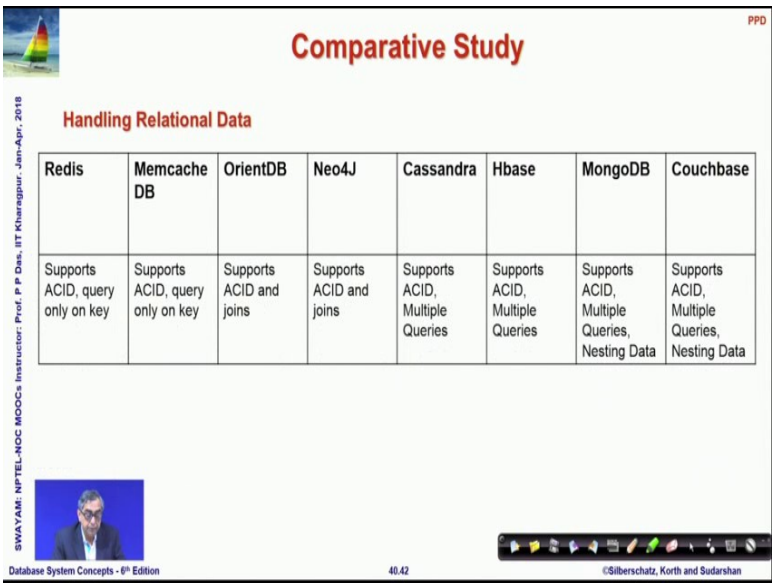
So, if you are interested, you can take specific courses which deal with the big data and prepare yourself for the bigger challenges ahead.

(Refer Slide Time: 26:29)



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.

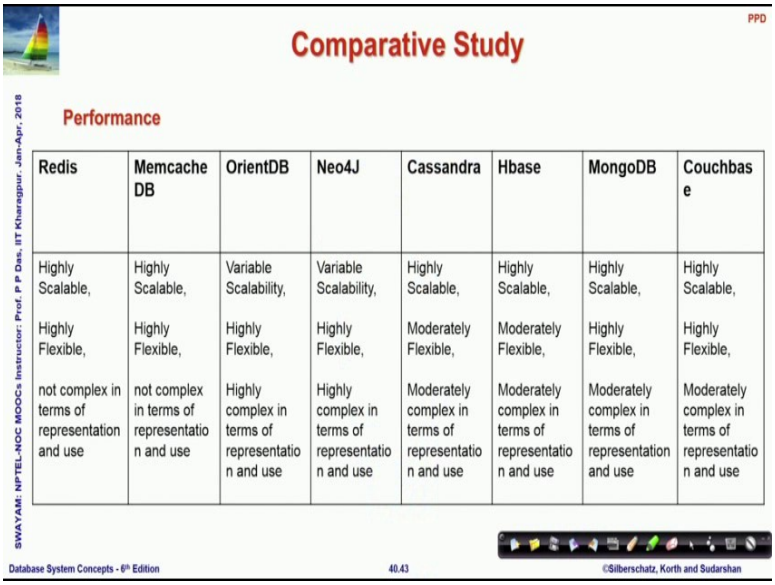
(Refer Slide Time: 26:46)



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?

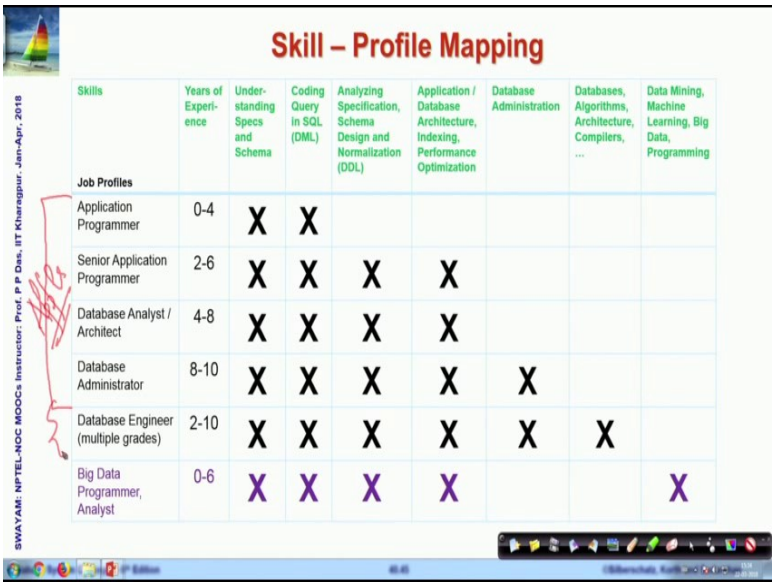
(Refer Slide Time: 26:59)



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

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)



Job Profiles	Skills / Years of Experience	Understanding 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
Application Programmer	0-4	X	X					
Senior Application Programmer	2-6	X	X	X	X			
Database Analyst / Architect	4-8	X	X	X	X			
Database Administrator	8-10	X	X	X	X	X		
Database Engineer (multiple grades)	2-10	X	X	X	X	X	X	
Big Data Programmer, Analyst	0-6	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.

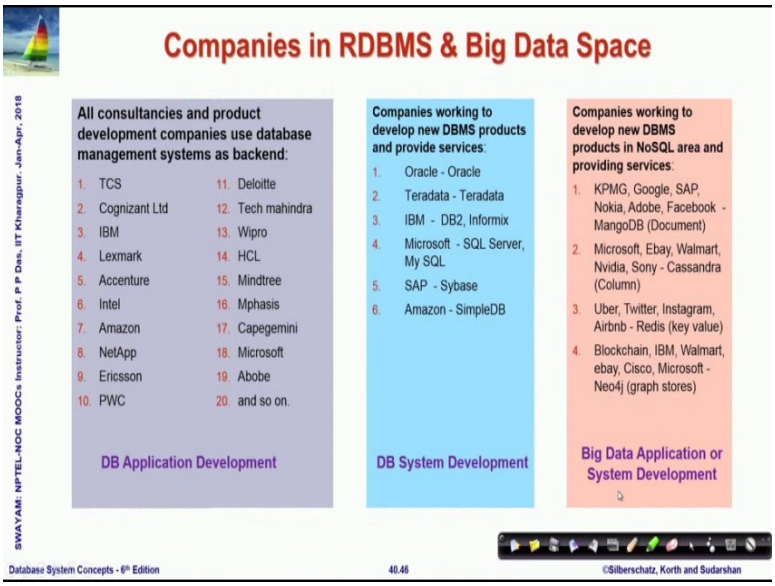
So, between these two, there will be certain overlaps a senior application programmer in addition to doing this might do some of these optimization techniques depending on how competent he or she is. Or some database architect may focus only on this, but these are the typical skills that you need. But 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

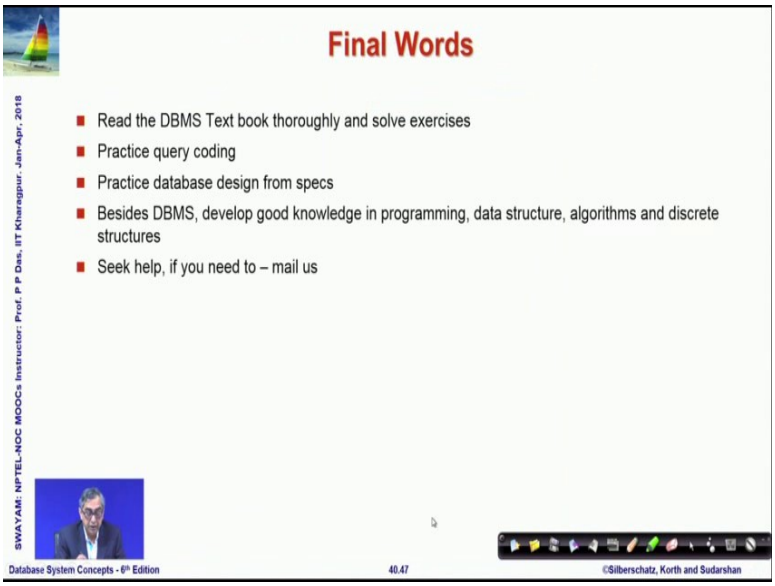
projects, product development different database management back end services and so on. So, DB application development, I have listed some around 20 companies, but there are really 100s of them almost. Any big organization in any area you think of, they require databases. So, in terms of beta based 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)



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.