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

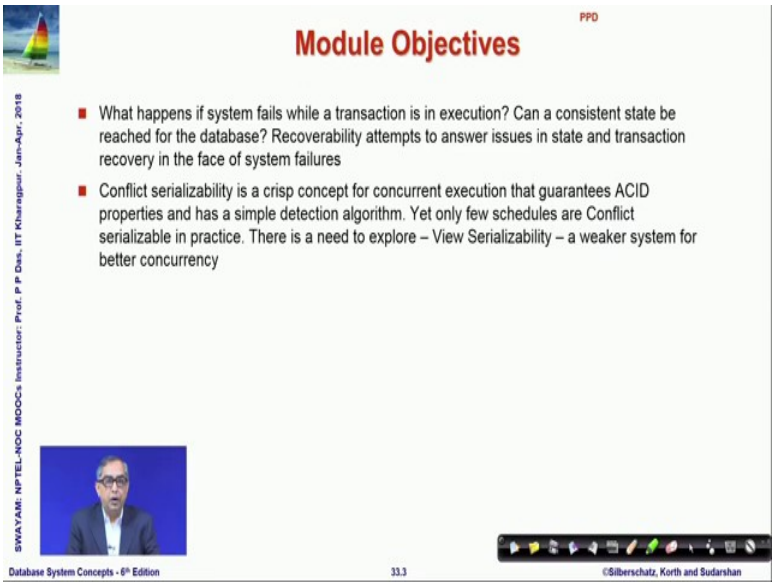
Lecture – 33
Transactions/3 : Recoverability

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

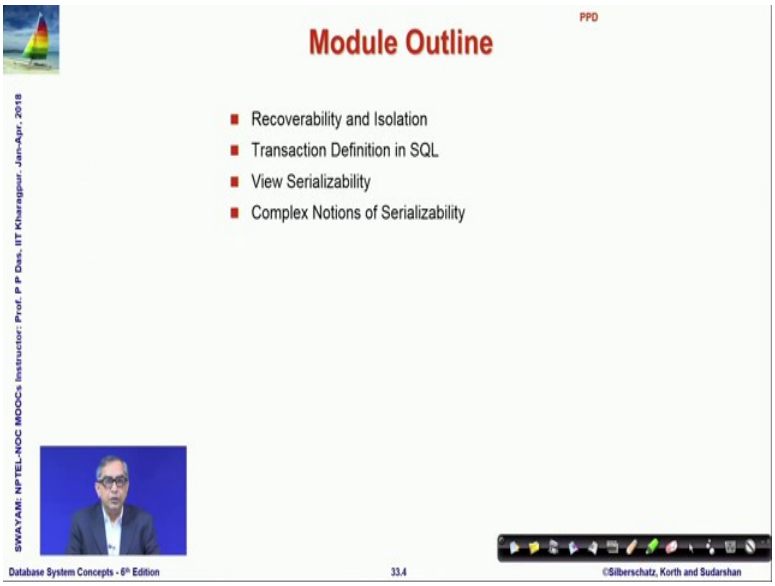
(Refer Slide Time: 00:50)



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.

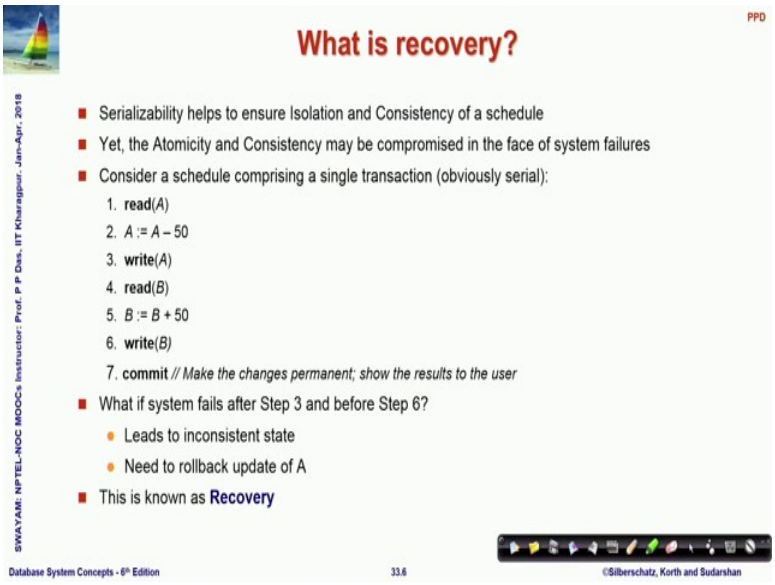
We would also look at that going forward from conflict serializability, what are the other notions of serializability, that can be used to serialize transactions and we will look at a weaker definition of serializability known as view serializability, which can serialize more schedules than what conflict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



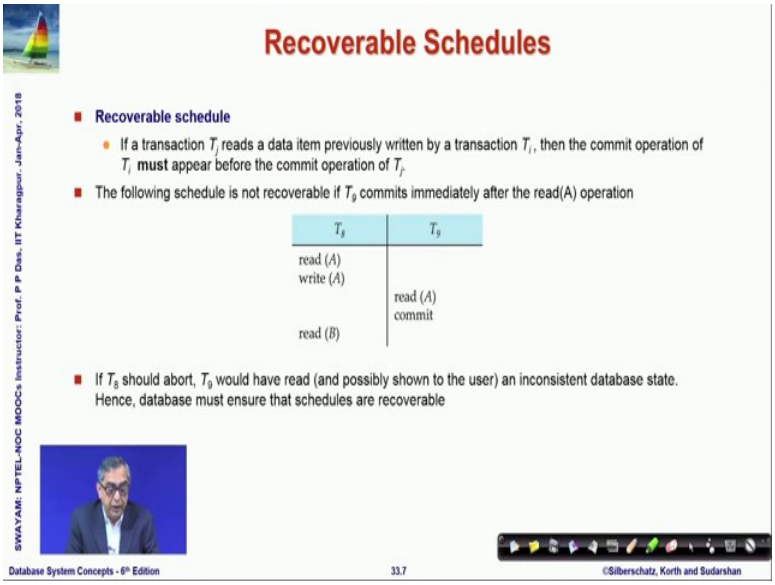
So, what we have done is we have seen the serializability help us, if we think in terms of the acid properties that we started by defining as the desirable properties of the transactions, we have seen that the serializability significantly helps us to achieve isolation and consistency of a schedule,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



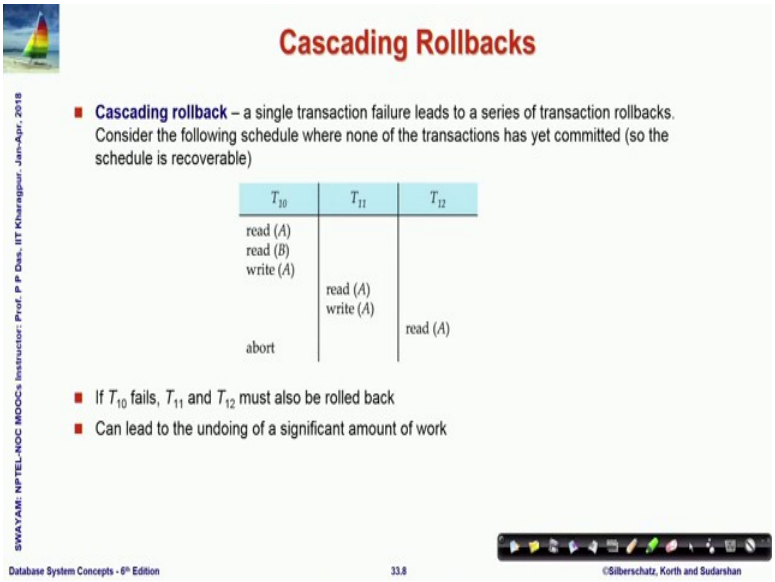
So, we say that a so, let us define a schedule to be recoverable if a transaction T_j reads A data previously written by a transaction T_i, then the commit operation of T_i must appear before the commit operation of T_j, if that happens then that is the earlier transaction which has written the data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

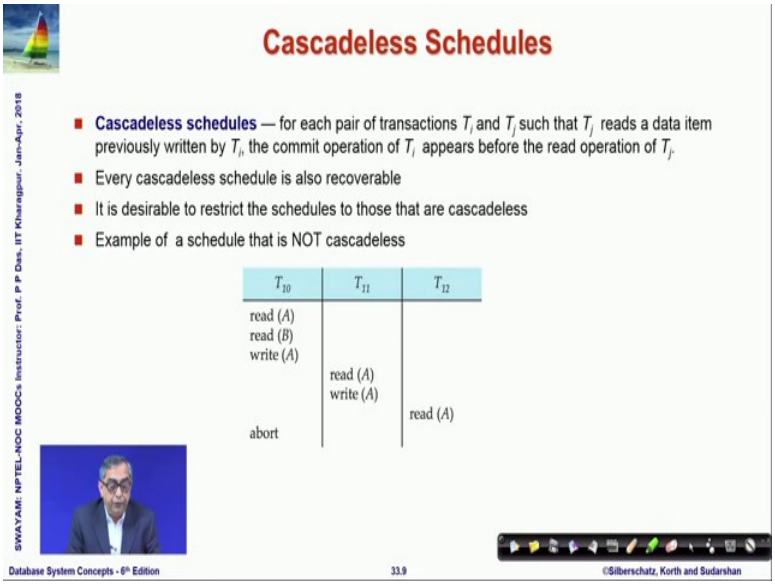


So, this is an example of a schedule which is not recoverable. Now let us also observe that a single transaction failure not only means that one transaction needs to be rolled back, but it could have a cascading effect, that is a series of transaction may require a rollback. So, here is an example of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

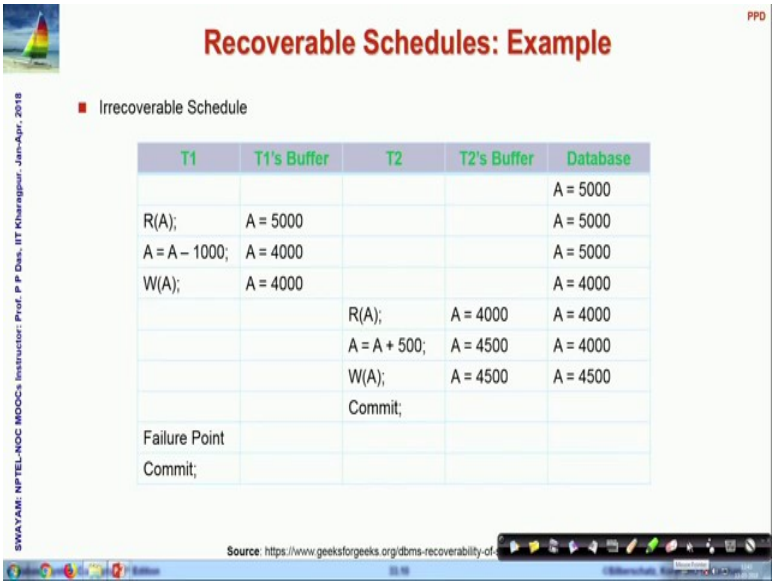


So, what we would prefer is if we could have schedules where such cascading roll back is not required. So, and there is a there is a condition through which you can achieve that. So, if we have a pair of transaction T_i and T_j. So, that T_j reads A data item previously written by T_i, then the commit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

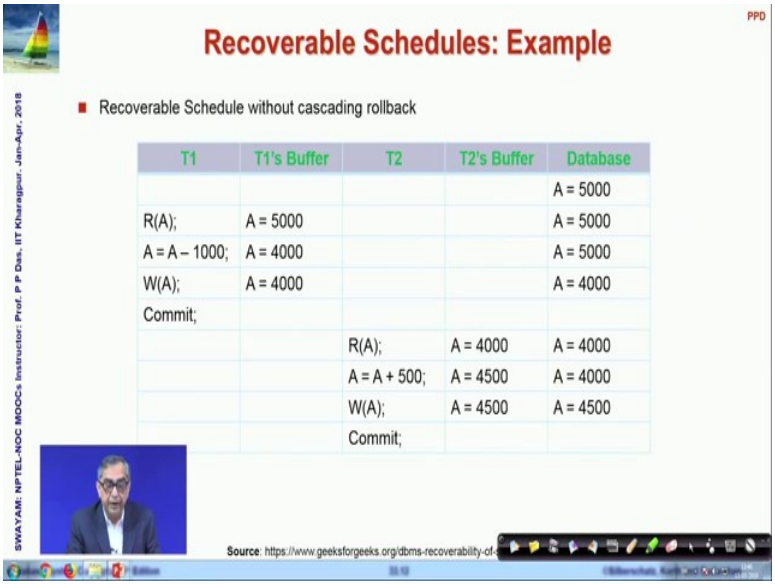
So, what will happen is read here and this value is a different A this is in the buffer or the memory of T_1 transaction, where A becomes 5000, then you subtract 1000 and then you write back the moment you write back in the database in between the value in the database is not changing, it is only that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

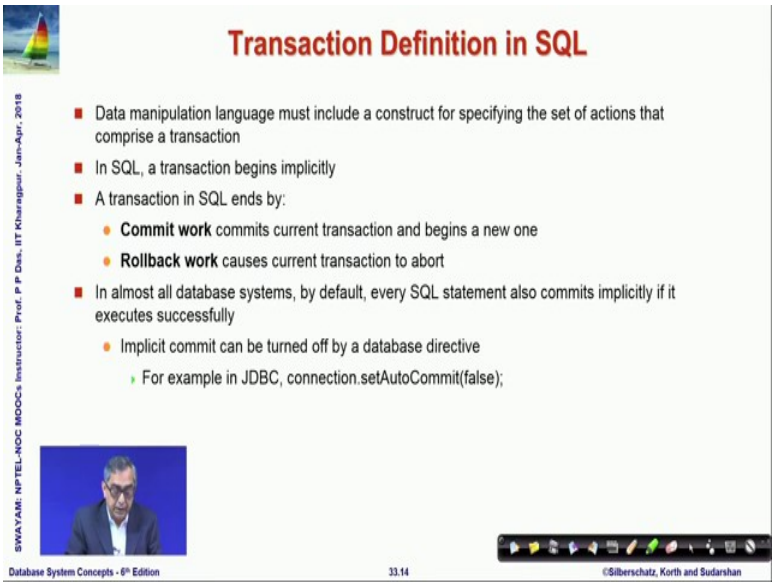
So, the transaction it is reading from it is actually committing the changes after that source transaction has committed. So, that satisfies the condition of recoverable schedule. So, you are able to recover, but it still required the cascading because T 1 had read A value in here of A which was not 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.

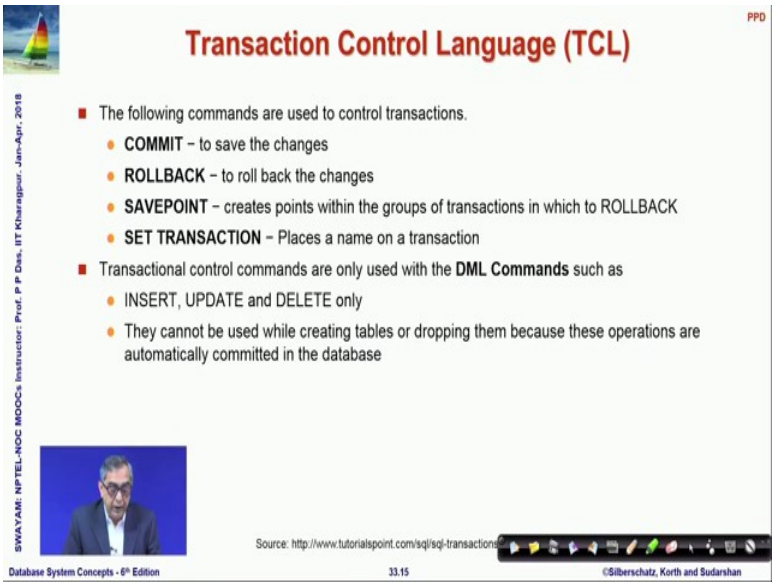
(Refer Slide Time: 14:12)



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.

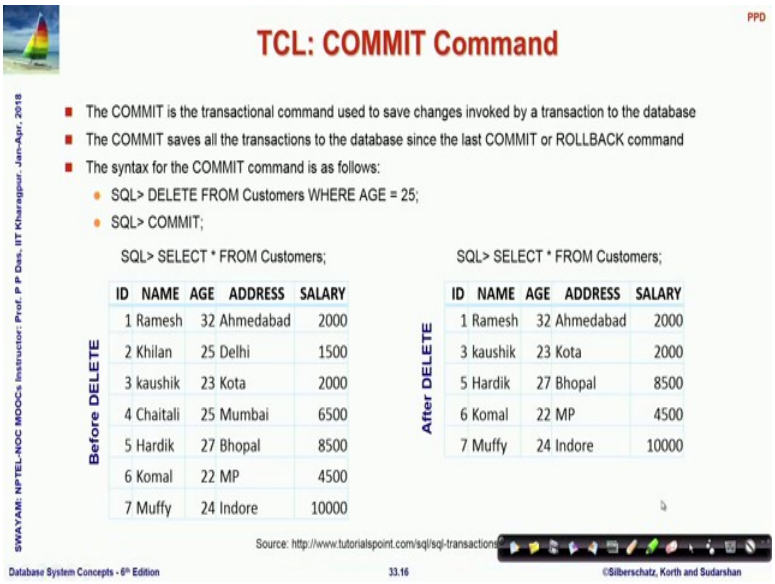
(Refer Slide Time: 15:22)



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.

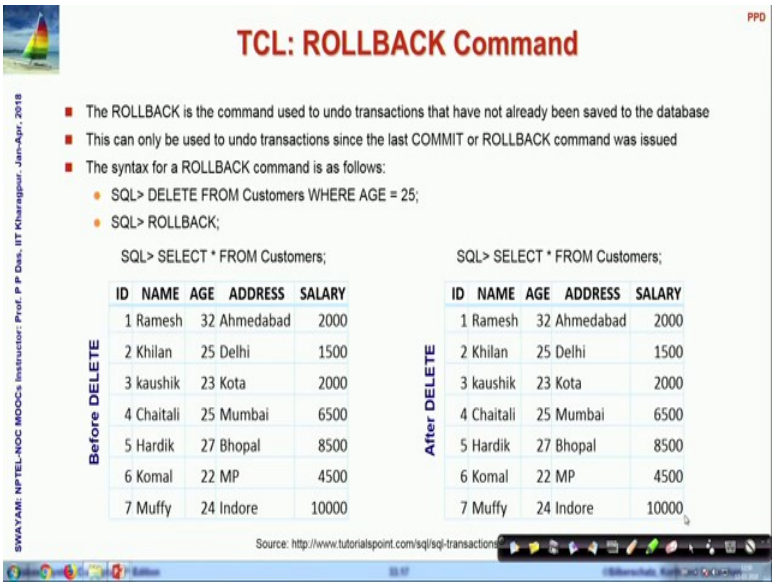
(Refer Slide Time: 16:09)



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.

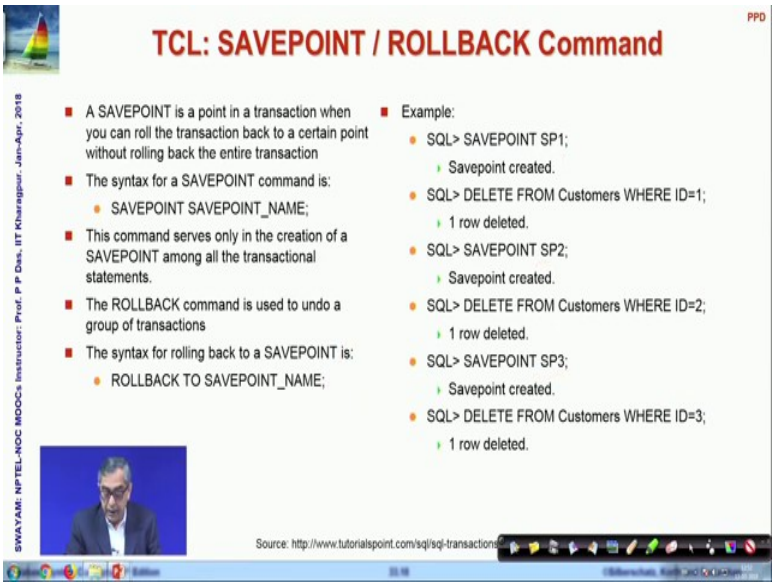
(Refer Slide Time: 17:11)



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)

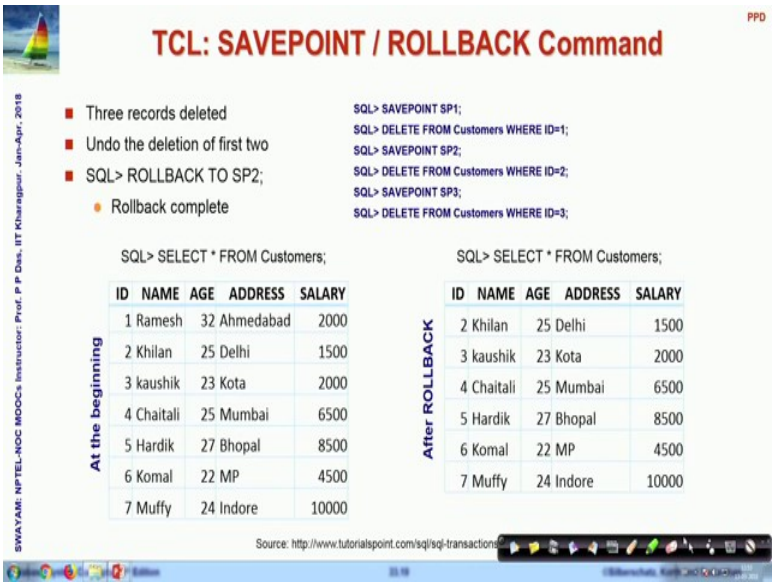


Now, you can a transactions often could be long. So, within the transaction you may want to mark certain points. So, that in case you roll back or you need to roll back, you can roll back to that particular point and those points are in the transaction are known as the **SAVEPOINT**. So, this is the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

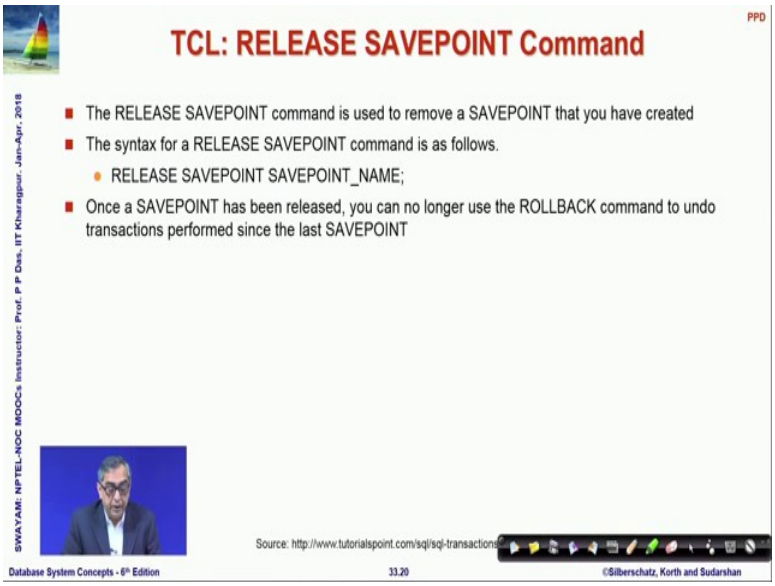
(Refer Slide Time: 20:14)



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.

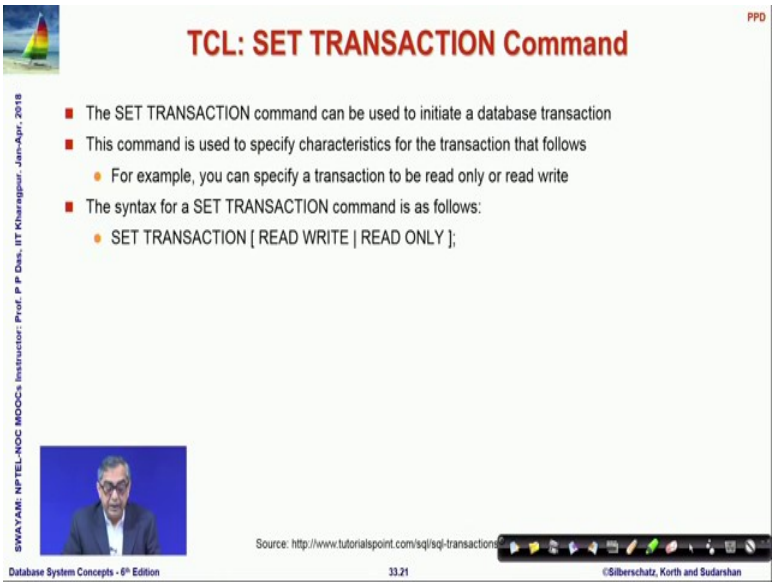
So, then when I undo the deletion of the last 2 records, then the what I see is the records which are marked as ID 2 and ID 3, which were done after SP 2 was marked which were deleted after SP 2 are marked, they are back into the table whereas, the deletion of SP 1 is still in effect and therefore, deletion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

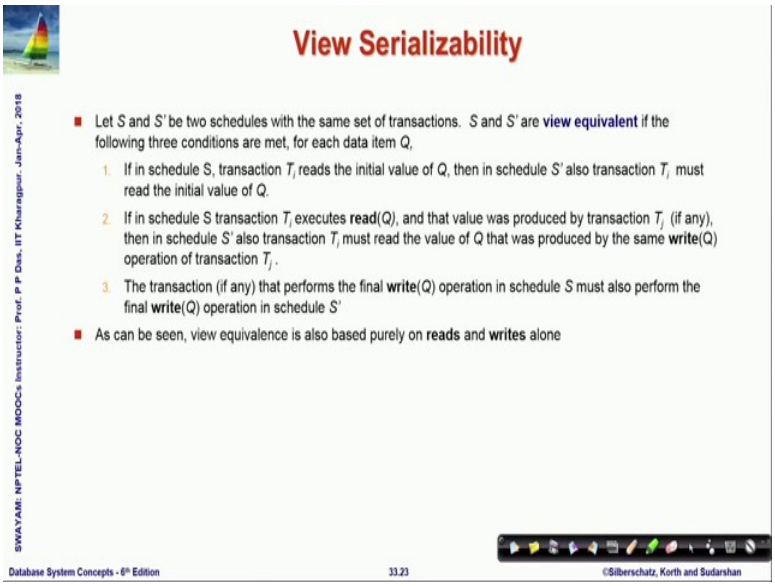


You can use **SET TRANSACTION** command to initiate a database transaction also and, it is typically used to specify the characteristics of the transaction, particularly if you want to say whether a transaction is a read only transaction or a read write transaction,

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.

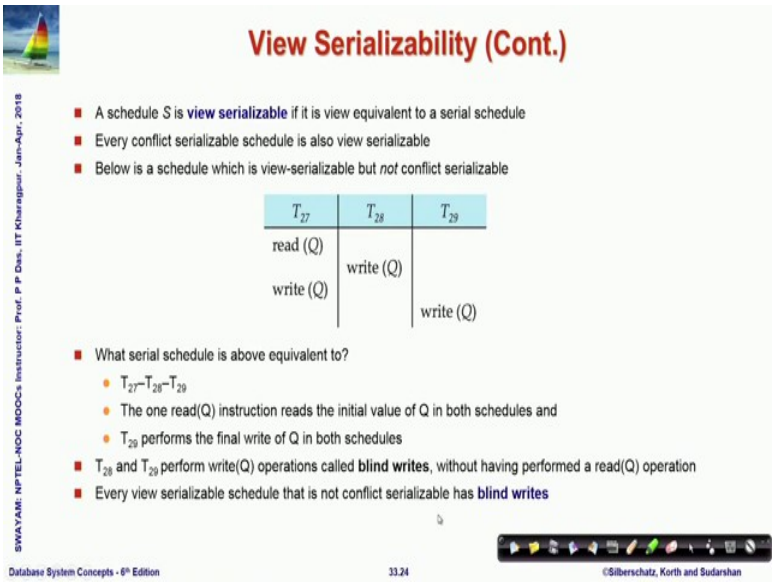
Similarly, the third condition says that the final write that is done, final value that it writes every transaction writes in both the schedules must be the same the same writes should operate. And the second conditions is a read write pair that every transaction when it performs a read on the data item,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

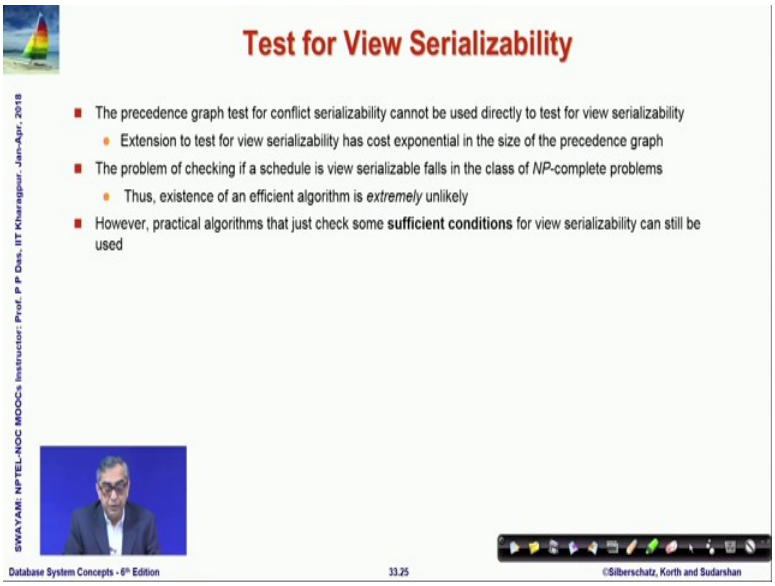
So, here is a schedule which is view serializable, but it is not conflict serializable, you know this is not conflict serializable because, certainly you cannot make it into a serial schedule make it equivalent to a serial schedule because, you cannot move this right Q above the right Q of T_{28} or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



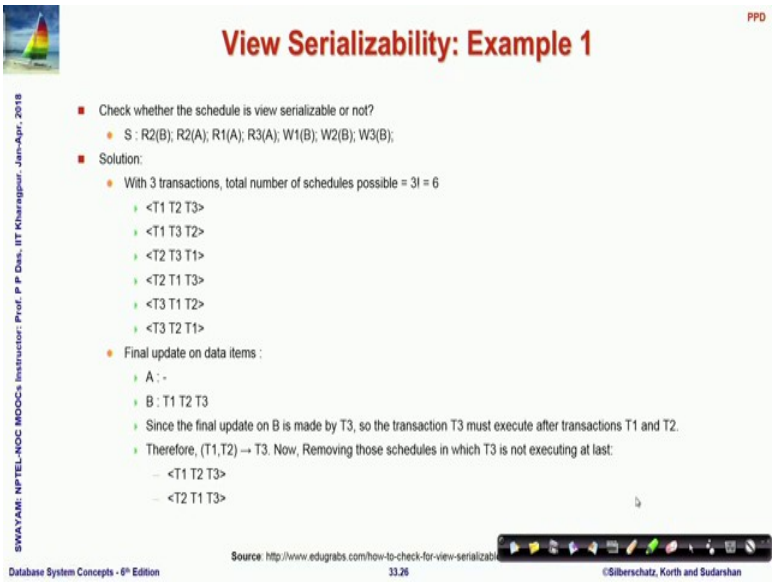
Now the question is similar to conflict serializability, where we saw that it schedule can be conflict serializable, if it is corresponding precedence graph is a cyclic. So, we would

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)



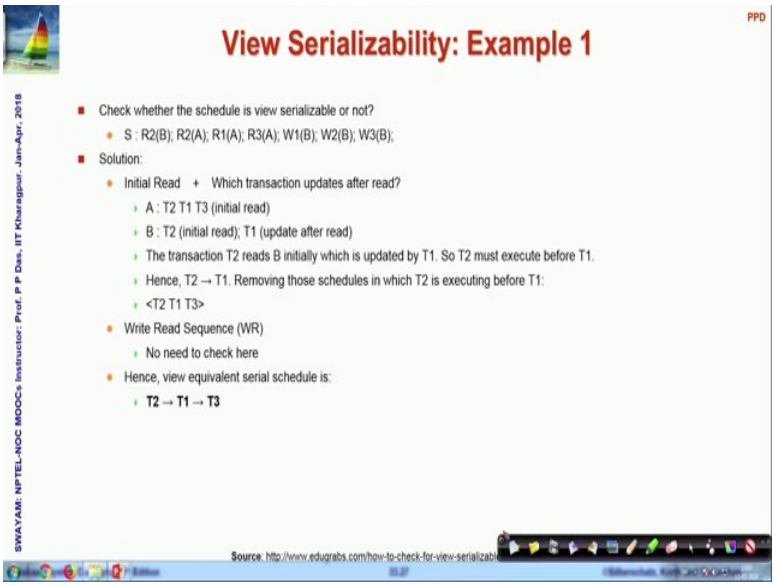
So, using view serializability have certain problems. So, here I have worked out a longer problem in terms of the view serializability to check that. So, it is kind of a brute force algorithm. So, if you see this is the schedule given there are two data items A and B and

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.

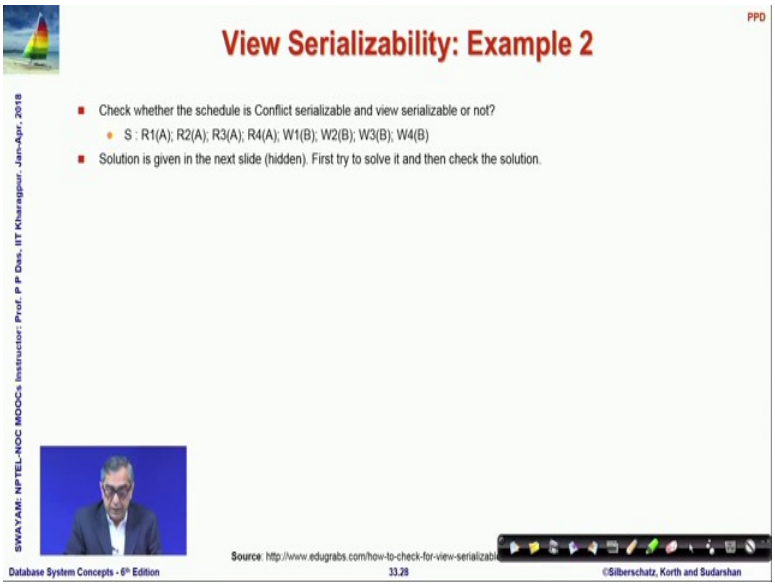
So, condition one checks that they must read the same value in both the schedule. So, we see that these are the reads that are happening on A. So, we see that on a there are reads happening, I am sorry this is these are the three that is reading A. So, it happens in the order of T_2 T_3 T_1 and T_3.

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.

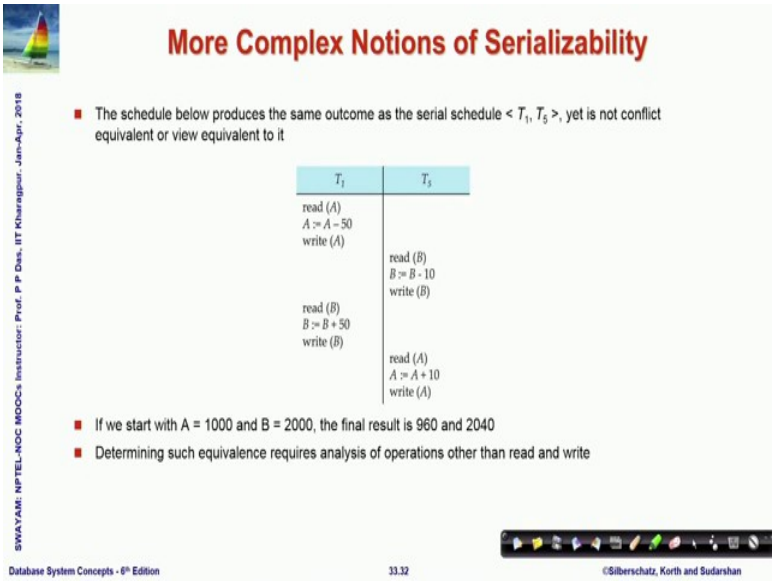
(Refer Slide Time: 31:45)



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)



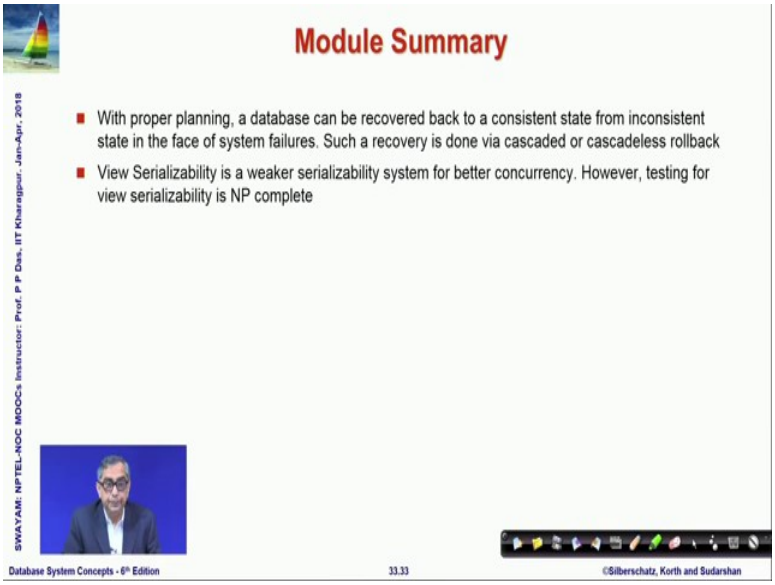
There are different other complex motions of serializability also for example, if you look at this particular schedule this actually is a serializable schedule, this is the effect that it produces will be same as the serial schedule of T1 T5, but if you go through the definitions of conflict equivalence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