

Database Management System
Dr. S. Srinath
Department Of Computer Science and Engineering
Indian Institute of Technology Madras

Lecture No. 27
Introduction to Transaction Recovery

Hello and welcome. In this session today we shall be starting with a new topic namely that of recovery that is how to recover data or how do we bring back database into what we called as a consistent state in the phase of any kind of failures. Failures could be of any kinds. Let us say disk crashes, power shutdown or network connection failure, many different kinds of failures and we are going to be somewhat specific when we say recovery. Obviously we cannot recover data that involved that were being processed in main memory during ram. Main memory during the crash but only we can recover whatever has been returned on to persistent storage like disc. But what is written on the disk? Good recovery or somewhat semantics associated to what is written on to the disk and do we have to do something more when in order to recover from crash schedule?

In order to answer this question we need to know the concept of a transaction in a database processing environment. Therefore, the title of session is called introduction to transaction recovery. In fact we are going to define the notion of correct recovery from wrong recovery based on the concept of transactions. We are not going to study about transactions in detail here. They are covered in a separate topic under itself transaction processing itself is a vast topic with several different aspects to it and we shall be concerned mainly with the recovery aspect here when it comes to and we are using the transactions to help us guide in deciding which kind of recovery is correct recovery which is incorrect recovery.

Let us first define the term OLTP. I am sure you might you heard of the term OLTP in several different contexts. It stands for online transaction processing and environment that is the database system plus an application program plus any other associated accessories like networks and so on that goes into form an environment that is meant for online transaction processing.

(Refer Slide Time: 3:22)



What is meant by online transaction processing? As the name suggests, it is a processing environment that can interactively process database transactions. We will define these terms in a much more accurate fashion later on.

(Refer Slide Time: 3:53)



But, let us first look at some environments that can be classified as online transaction processing environment. Some examples are like airline or railway reservation systems. What are the characteristics of such system? One of the main characteristic that you can straight away see in Railway Reservation Systems, for example:

(Refer Slide Time: 4:29)



Is that there are several numbers of users who are accessing the database system simultaneously that is you might have experienced it if you have tried booking a railway ticket over the internet. Suppose a train is getting almost full and if you delay in booking your ticket given by let us say some times even by few minutes, then never you may not get a conform ticket at all might go in to waiting list.

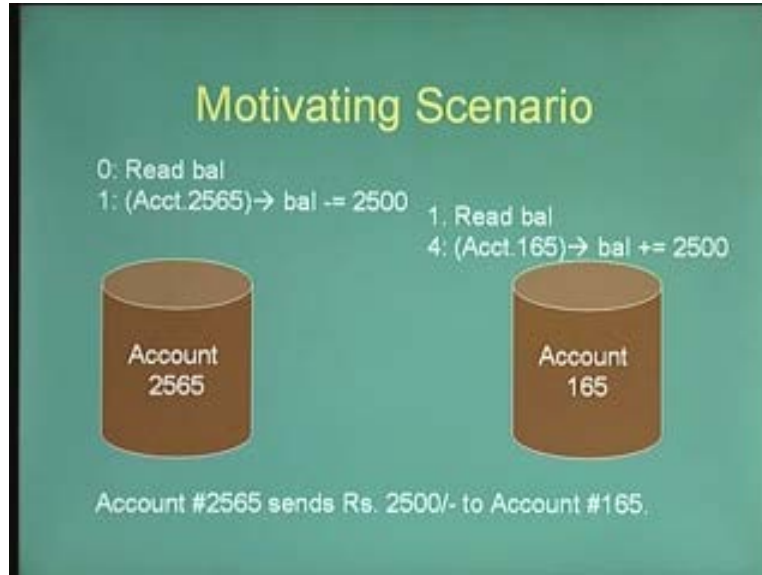
So that means, at that particular instance of time when your checking the status of tickets, there were several other people accessing the same database throughout the country. That is they could be accessing via the internet, could be accessing via let us say some kind of Queuing system that is across the booth or whatever from, they are all accessing the same database and several different transactions are happening at the same time and similar examples are of that of banking systems and especially how cash dispensed in atm's or wire transfer mechanisms and so on. Another over tip environments require super market checkout systems and where customers come in with their baggage of whatever things that they have bought, they have to be checked out praised and build and so on.

(Refer Slide Time: 5:52)



There are hotels and hospital system, trading and breakage system where buying and selling of the shares keep happening continuously whenever trading is on and several such sessions are happening simultaneously. Now let us take scenario which helps us understand what should be the properties of these transactions that go on in OLTP environment. Now consider a small banking example where different accounts are maintained in a bank. There could be on different databases or within the same database or different locations or same location, it does not matter. Let us just consider that there are different accounts and this database is being accessed simultaneously by several users performing several transactions. Let us say that one percent using net banking to transfer some money from his account to somebody else account.

(Refer Slide Time: 7:14)



At the same time, other person is using an atm to withdraw some money or deposit some cheque or some cheque is getting encashed some did is getting encashed or withdraw whatever several things are happening simultaneously in the bank. Now, among this let us consider a small one particular example that account number 2,565 send 2,500 rupees to another account number 165.

So let us say at time t equal to 0 however we define our time, a transaction begins or set of database operations begin the application program that is making this wired transfer will initiate database operations which will first read the balance of 2565. Now it will read the read the balance amount and because it is withdraw, it is a withdrawal from account 2500 will be deducted from this balance and let us say we are using concurrent applications which can have different threads of execution which can run at the same time.

So when this balance is being deducted here at the same time the balance of second account is being read of account number 165 and due to some reason, this thread process gets swapped of into disk by the operating system. Some other process is running because note that operating system of the scheduling processes to and some other processes running at the time and by the time, this process comes back. It is time number 4, time t equal to 4. At this time, balance amount of 2,500 is added to this account.

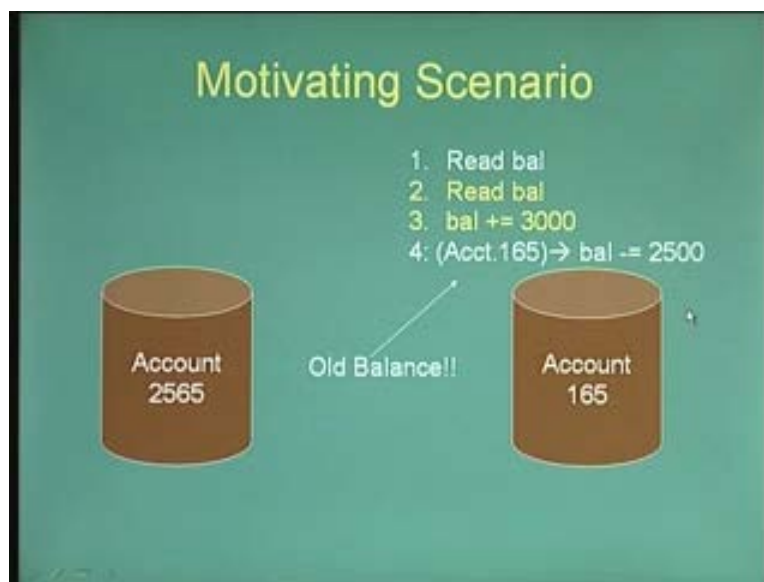
Now meanwhile let us say there is another transaction. Let us say, the account holder of this of this account number 165 is meanwhile standing in an atm and depositing certain cash certain amount of cash to the atm or may be he has sent a cheque and that cheque is getting clear. So, why this person with the account 2565 performing wires transfer? The person having the account 165 is also depositing amount of 3,000 rupees and it is so happens that, the way processes are scheduled this set of operation that is reading the

balance of account number 165 here and adding 3000 rupees is done before the previous transaction finished.

That is before the balance of before amount of 2,500 is deducted from that is based on the previous transaction. You can see what happens what has happened now? The previous transaction has read the old balance amount and added. Actually this should be added. I am sorry. There is a small bug here. This should be plus equal to that it is taken. It is taken previous balance amount and added to 2500 rupees here. While before it could that the previous balance was read, 3000 rupees was added by the customer who is depositing his cheque from the atm.

Now what has happened here in that, this entire transaction is lost this entire serious of operation is lost after time t equal to 4 because suppose this person had 50,000 rupees in his account it will be now 52,500 rupees rather than 55,500 rupees. 3000 plus 2500 rupees.

(Refer Slide Time: 11:12)

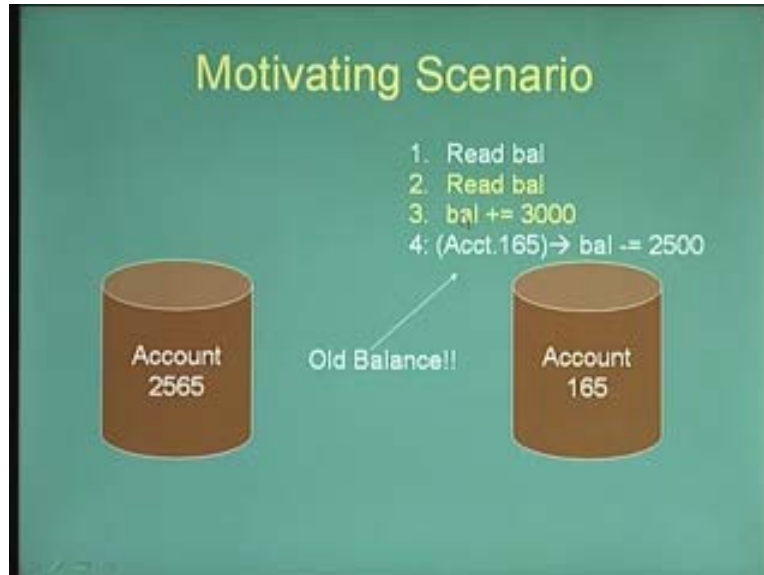


So 3000 rupees is just gone, it is just lost. Now, as you can see this is not unrealistic situation especially we just saw today, when we have facilities like net banking or atm or booking train tickets over the web or using telephone calls or using sms from mobiles and so on. This is not an unrealistic situation because concurrency is concurrent activities are happening at the same time. I mean a concurrent activity is happening all the while and if you are not careful, such kind of activities can result in an inconsistent database. The entire transaction of depositing 3,000 rupees is lost in this example.

So, if you are not careful what is the clear you have to take what is that you need to remember when we are dealing with situations like this. The thing that you need to remember here is the first two activities. Let us say the first activity of wired transfer

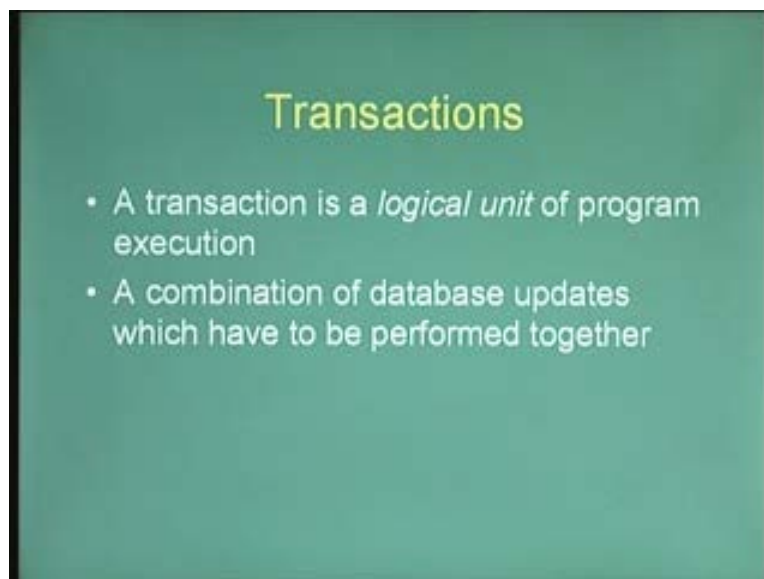
between account 2565 165 is a completely different or is a conceptually separate or distinct activity from the second activity of depositing 3000 rupees.

(Refer Slide Time: 12:47)



So the first activity is a conceptually or logically separate activity or functionally than the second activity. Such kind of logical units of works are called transactions and transaction activities in a transaction, the actual database activities transaction should be schedule in such a way such that these kinds of anomalies do not occur. Let us try to formulate these things in a little bit in much more detail. Now what is a transaction? A transaction is a logical unit of program execution.

(Refer Slide Time: 13:35)



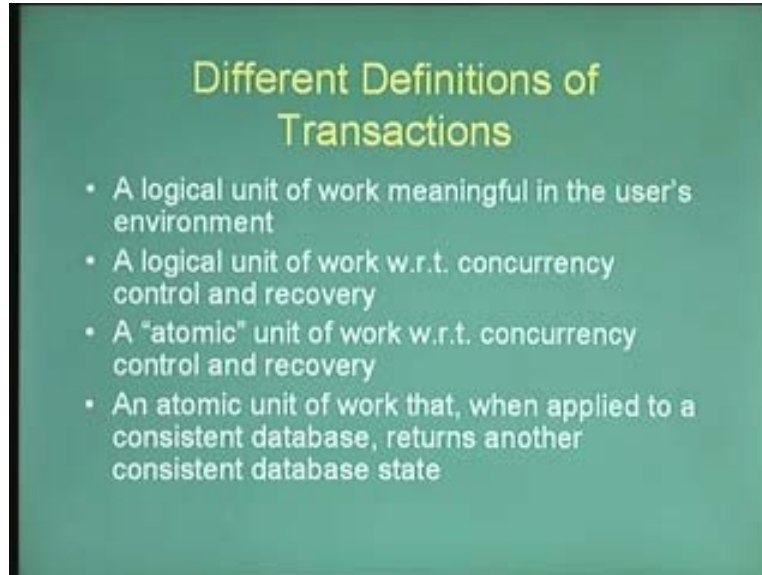
As we saw the entire activity of withdrawing money from A's account and depositing that money in to the B's account constitutes one logical unit of operation and it is a combination of database updates which have been performed together. They cannot be independent of one another. The withdrawal from a's account is not independent of deposit into b's account and vice versa. There are several different transactions depending on where and how it is being used.

Let us have a brief look at the different definitions of transactions which makes it clear what are the different tests to handling transactions or recovering from transactions one. Firstly we can define a transaction as a logical unit of work that is meaningful in the user's environment. As we can see here, the wire transfer that is withdrawal of money from A's account and depositing in to B's account is a meaningful semantic activity as part of the users environment because that constitutes a semantic process in the users environment, a wire transfer. Similarly depositing 3,000 rupees; In order to deposit 3,000 rupees, there were two database operations. Network done that is reading previous balance and updating the balance. So both these activities of reading and updation is one semantic activity that, it constitutes one meaningful activity namely that of depositing certain amount of money in to an account.

Now one can define even transaction as a logical unit of work with respect to concurrency control and recovery. Not necessarily semantic activity in terms of the user's environment. Many cases users depending on what granularity you are looking at. Users may not be concerned with water considered transaction set at the database level. However, we might have to club or we might have to combine certain database activities in to transactions in order to maintain consistency in the phase of concurrency control and recovery process. Transactions are also called atomic unit of work. Instead of calling it as logical unit of work, much more stringent definition is to say that it is an atomic unit of work with respect to concurrency control and recovery. Atomic unit of work is a more stringent requirement than saying logical unit of work. An atomic unit of work basically means that it cannot be subdivided in to smaller works.

Either all of the activities of a transaction are performed or none of them are performed. You cannot perform half a transaction and leave it at that or you cannot perform 90 percent transaction and leave it. Either you have to perform the entire set of activities of a transaction or nothing at all or another definition that it is generally used that is transaction is an atomic unit of work that will apply to consistent database returns another consistent database.

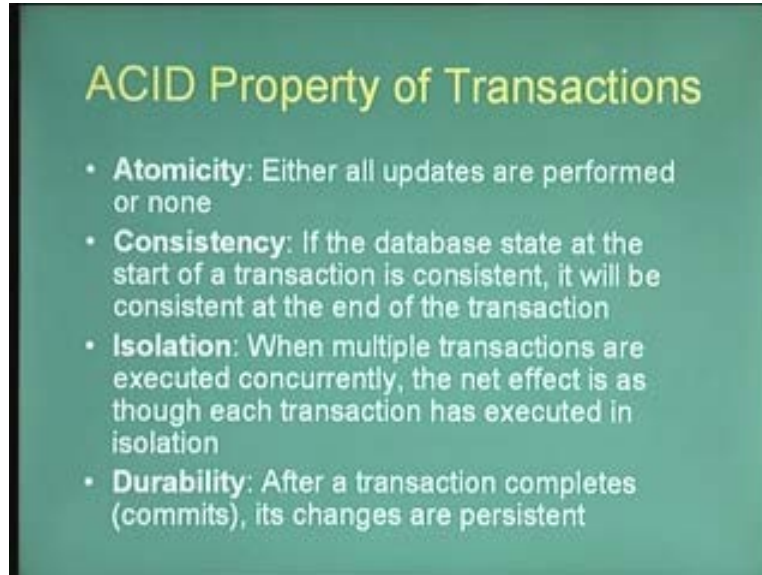
(Refer Slide Time: 16:58)



That is an atomic unit of work that is meaningful, that could be meaningful that is the atomic unit of work with respect to concurrency control recovery. However, not all atomic units of work that can be managed for concurrency and recovery could be transactions because some of them could take it take the database to inconsistent state. So the transactions sometimes defined as in those only units of work make transactions to a valid state of database. Now what are the properties that a transaction should satisfy that, we have seen we have motivated the need for transaction in several different from angles.

We first saw an example application an example OLTP application where transaction processing incorrect transaction lead to anonymous and also we saw different definitions that look at the notion of transactions from different levels. Now how can we consolidate them together and synthesis what are the basic properties the transaction should hold? So the basic properties that a transaction should hold are called as the acid properties of a transaction. Acid stands for atomicity, consistency, isolation and durability that is shown in the slide here. So, what is atomicity in the acid property? Atomicity we just saw in the previous slide that a transaction should be viewed as an indivisible unit of work.

(Refer Slide Time: 18:46)



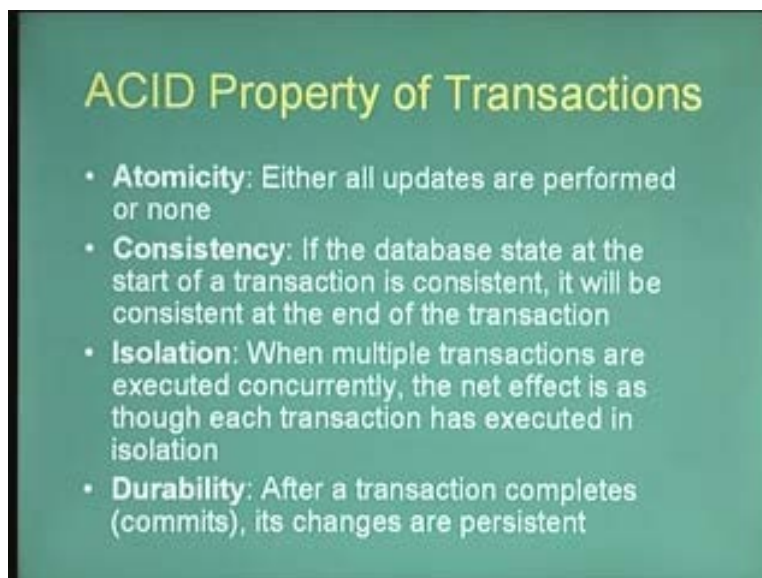
A slide with a green background and a black border. The title "ACID Property of Transactions" is at the top in yellow. Below it is a bulleted list of four properties: Atomicity, Consistency, Isolation, and Durability, each with a brief definition.

ACID Property of Transactions

- **Atomicity:** Either all updates are performed or none
- **Consistency:** If the database state at the start of a transaction is consistent, it will be consistent at the end of the transaction
- **Isolation:** When multiple transactions are executed concurrently, the net effect is as though each transaction has executed in isolation
- **Durability:** After a transaction completes (commits), its changes are persistent

That means either all activities in a transaction should be performed or none of them. We cannot perform half a transaction and leave it. Consistency: Consistency of a transaction basically means that of the database is consistent transaction should be consistent after a transaction. The transaction the atomic unit of work should not lead the database in to an inconsistent state. What is meant by an inconsistent state? Any state that violate the integrity constraint of the database.

(Refer Slide Time: 19:17)



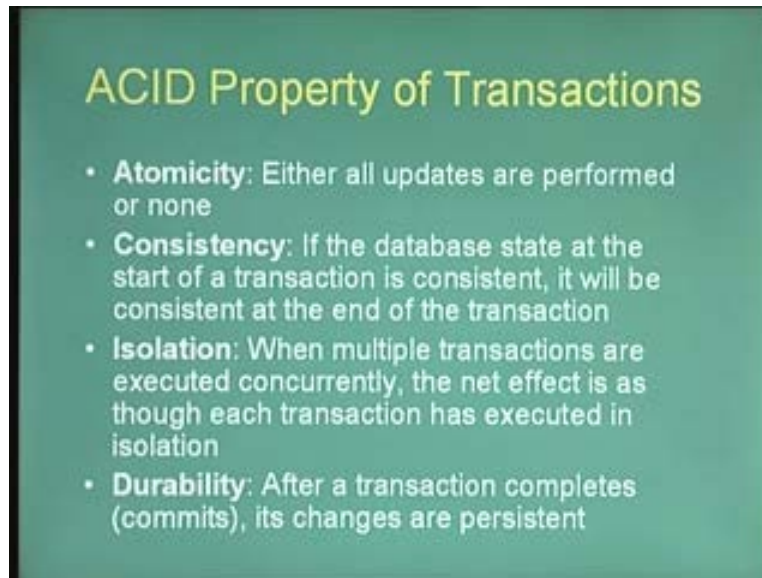
A slide with a green background and a black border. The title "ACID Property of Transactions" is at the top in yellow. Below it is a bulleted list of four properties: Atomicity, Consistency, Isolation, and Durability, each with a brief definition.

ACID Property of Transactions

- **Atomicity:** Either all updates are performed or none
- **Consistency:** If the database state at the start of a transaction is consistent, it will be consistent at the end of the transaction
- **Isolation:** When multiple transactions are executed concurrently, the net effect is as though each transaction has executed in isolation
- **Durability:** After a transaction completes (commits), its changes are persistent

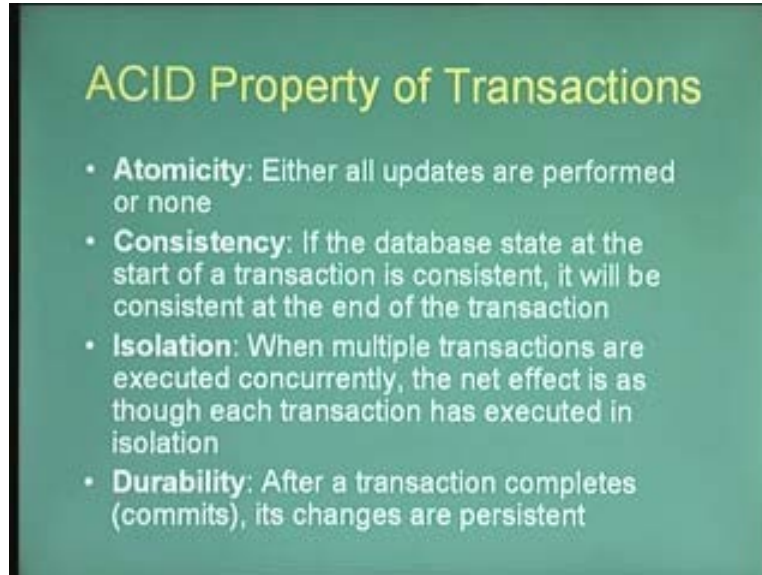
We saw how to specify integrity constraints and how they are enforced in a database system. The third property of transactions is isolation. We saw an example of transaction violating isolations in our banking example that we saw before. Isolation essentially means that even though activities in the database are happening concurrently that is the readings and updates and reads and writes operation whatever is happening on to the database level are all happening in a simultaneous fashion.

(Refer Slide Time: 19:53)



The net effect in an OLTP environment, the net effect should be as though the transactions have been executed in some serial order. It does not really matter to what should be the serial order as long as we can establish equivalence between the way in which database activities are performed to a serial sequence of execution. That is it should be as though that transaction 'a' was completed before transaction 'b' begin. 'b' was completed before transaction 'c' begin and so on and the last property of the transaction is called durability, that is once the transaction finishes are rather we used the term commit here, that is once the transaction says now i have done all my work and you can commit whatever changes have made in to the database. Once it is committed, the changes are persistent.

(Refer Slide Time: 20:52)



You cannot rollback or you cannot undo the changes that are being made by the transaction after it is being committed to the database that is commit is something like in order to understand the notion of commit, it is something like a physical activity. For example: dispensing money from an atm is a commit operation. Once it is committed, once money is dispensed, you cannot rollback. You cannot expect the user to say no no no, we did something wrong. We have to put back money that i gave you because some other transaction is conflicting. So once commit operations is performed, it is durable that is the transaction cannot be rolled back and we have to do something else in order to undo the operations of transaction.

Once you dispense the money from an atm you have to do something else. We chased the person who withdrew the money and get back from him if required and so on. So we cannot undo the transactions within per view the database. Let us look at the examples that specify each of these acid properties of the transaction. Have a look at slide here. Let us say the transaction involves again a wire transfer from account a to account b that is, the transaction should be like this. Account a dot balance minus equal to amount whatever amount has to be transferred and account b dot balance plus equal to amount. Let us suppose that the balance that is amount number of rupees has been debited from a's account and for some reason the databases crashes.

Let say there is disk crash or a network failure whatever or operating system crash or whatever. Now once the system is brought up again that is once there is a recovery process, this transaction has not completed. Therefore in order to make it atomic, we have to roll back. We have to roll back the changes that you have made since the beginning of the transaction that is, we have to put back this amount back into a's account and then restart the transaction once again. Otherwise, it would not be an atomic operation.

(Refer Slide Time: 23:17)

Atomicity

Consider a case of funds transfer from account A to account B.

```
A.bal -= amount;
B.bal += amount;
```

A.bal -= amount;
CRASH

...

RECOVERY
A.bal += amount;

Rollback

This amounts to if it did not perform the recovery operation this would amount to performing half a transaction and we saw that performing half a transaction and we saw that half a transaction is not an atomic transaction. What about consistency? Have a look at the example here again. This is again the wire transfer example from account 'a' to account 'b' that is the same series of operations have to be performed that is A dot balance minus equal to amount and B dot balance equal to amount. Now let us say that the query that has to be performed. These two things have been that is the query planned has been performed and these two operations are given to two different threads in the operating system.

(Refer Slide Time: 24:05)

Consistency

Consider the case of funds transfer from account A to account B.

```
A.bal -= amount;
B.bal += amount;
```

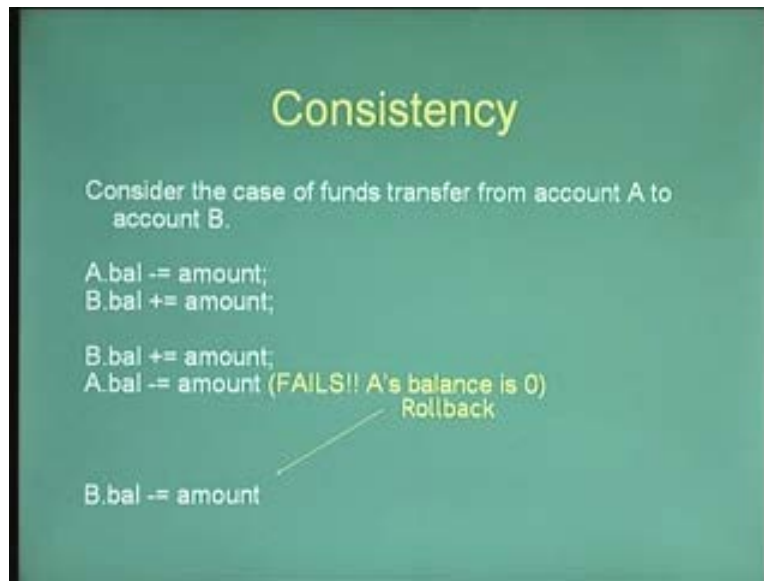
B.bal += amount;
A.bal -= amount (**FAILS!! A's balance is 0**)

Rollback

B.bal -= amount

And it so happens that the thread performing changes on B's balance is scheduled first before that of 'A'. Let us say first 'B's balance amount is crediting. Let us say 'A' is sending 2500 rupees to 'B'. So we know the amount 2500. So these balances added by a value of 2500 and however when trying to recover, remove 2500 rupees from A's account. We see that A has zero balance in his amount. He cannot make this payment. So this transaction fails. We cannot make this transaction. So in order to keep this transaction consistent, we have to deduct whatever credit we made in to the account of B in order to bring back the consistency in the database systems.

(Refer Slide Time: 25:06)



Note that here there is no crash or anything of that sort. Here there is the normative failure. The normative or failure with respect to nor. The failure which violated the integrity constraint. We can think of an integrity constraint that says odd raff are not allowed that is the balance amount in users account may never be negative. So when we try to do this operation that is when you try to debit 2500 rupees from 'A's account, we found that the balance is becoming negative and it violate the integrity constraint which in turn cause the transaction to roll back that is in order to maintain the consistencies in the database systems.

The third property is that of isolation and isolation like you said before deals with concurrency that is what how do we handle concurrent operations being performed from two or more transactions simultaneously. So again consider the case of wired transfer another case of wired transferred that is account A's is transferring some account to B. At the same time, account the person holding account 'A' is also withdrawing some money that is the person holding account A has given a check at some time which is getting process now and if the same time, the account holder is withdrawing some money. Let us say the transaction t1 is reading account A's balance debiting the amount and crediting the amount account B.

Let us say that A's balance will become zero after debiting this amount. Let us say it 2500 rupees and let us say the same amount is also being withdrawn by being as by account holder for withdrawal. The net effect of running these two transactions should not be the case that both of them read the database or if the balance, there is 2500 rupees and then go ahead independently debiting them debiting the account because that would be in correct because we would have debited more than 2500 rupees.

From these two transactions where it is not could not be reflected. So the net effect should be T 1 precedes T 2 that is T 2 begins only operation after T 2 completed is in effect that that should be the case. Which case T 2 will fail or Tone begins operations after T 2 completed? It does not matter which is the serializable schedule which is the serialize schedule that we want is it T 2 after Tone after T 2. So in either case none of the 2 transaction will fail that is either the withdrawal will fail or the wired transfer will fail.

(Refer Slide Time: 28:02)

Isolation

Consider the case of funds transfer from account A to account B.

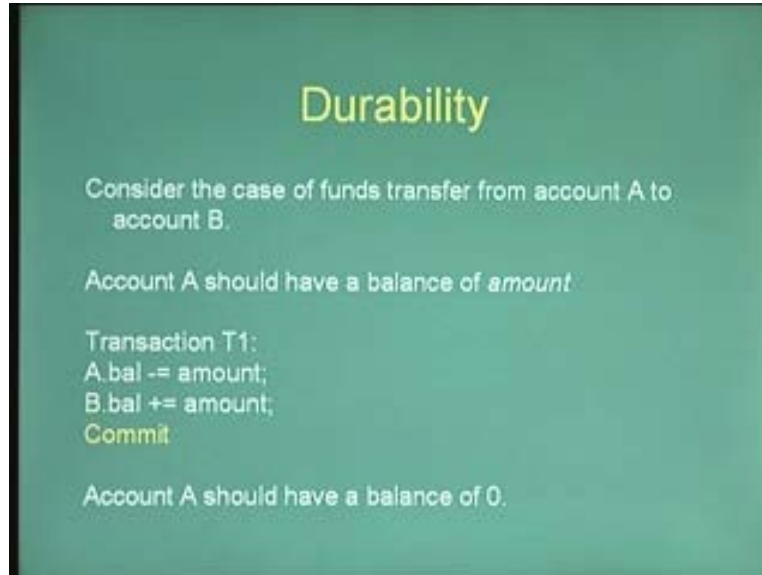
Transaction T1:
A.bal -= amount; (Let A's balance become 0 after this...)
B.bal += amount;

Transaction T2:
A.bal -= amount2;

Net effect should be either T1,T2 (in which case T2 fails) or T2,T1 (in which case T1 fails)

And durability like we said is the commit operation that is one thing committed, then it is not change we gave an example of let us say money dispense information from atm.

(Refer Slide Time: 28:15)



Durability

Consider the case of funds transfer from account A to account B.

Account A should have a balance of *amount*.

Transaction T1:
A.bal -= amount;
B.bal += amount;
Commit

Account A should have a balance of 0.

Once the commit operation is performed, it is safe to dispense money from the atm and we cannot roll back the transaction once the commit is performed. What are the different states in which a transaction is in and this is important to know when we are trying to recover from a failure of a transaction? Now transaction is set to be in several different states depending on what has happened since it begins. It is said to be in active set which is the initial set when the transaction is executed. When the last statement has finished execution and it is ready to commit, the transaction is said to be partially committed.

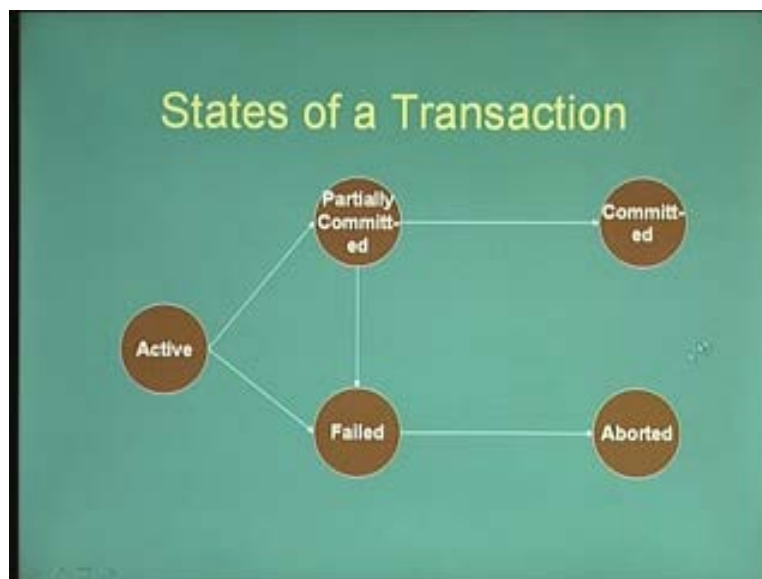
When the transaction discovers that it no longer proceeds with normal execution because something else has happened, some crash or some violation of an integrity constraint or some violation of an isolation requirement and so on. When it discovers something like that then it is said to be in the fail state and once rolled back, it is said to be aborted and if the transaction successfully completes that its operation that is an atm successfully dispenses money. It is said to be committed and either committed or roll back or aborted state is called terminated state.

(Refer Slide Time: 28:30)



So this slide schematically depicts the different states in a transaction and also shows from which state you can go to which other state that is from the active state, you can go to either a partially commit state or a failed state and also you can reach fail state from a partially committed state that is after performing a few operations and from a partially committed state, if everything okay then you can go to committed state or if the things are not okay you can go in to the fail state turns take in to an aborted state.

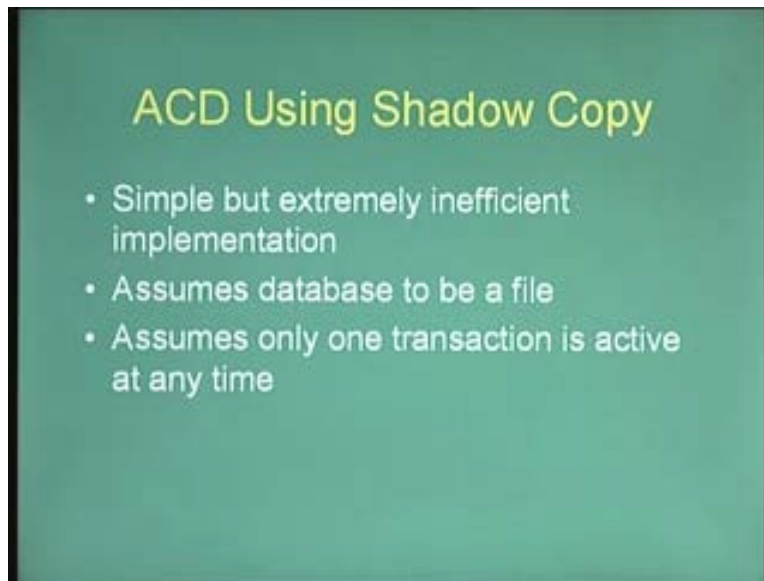
(Refer Slide Time: 30:14)



Let us have a simple look at how these acid properties can be maintained or what it takes to maintain these acid properties and we are going to look at simple example called

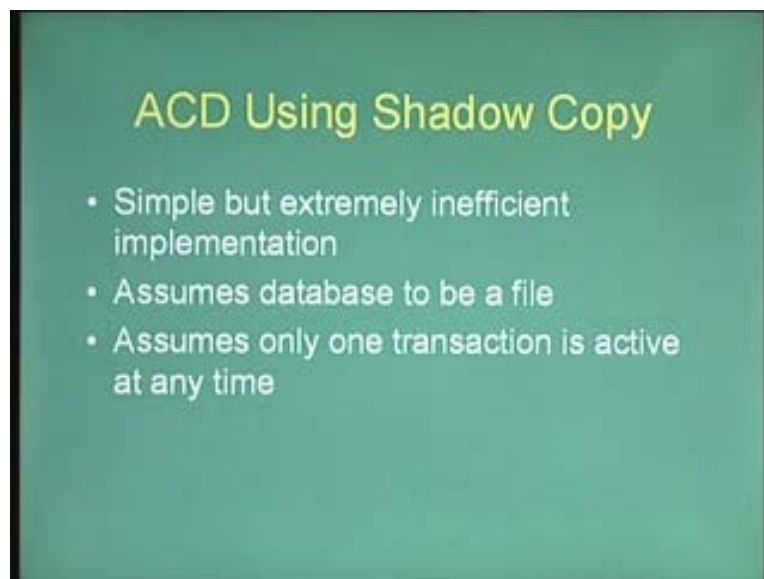
simple technique called a shadow copy. Shadow copy is extremely simple extremely inefficient and it is not used in practice. Several more sophisticated techniques for handling are maintaining acid properties or taken up in much more detail when we take up the topic of transaction processing itself. Here this is just to illustrate the concept of what it takes to perform, to maintain certain properties of a transaction.

(Refer Slide Time: 31:06)



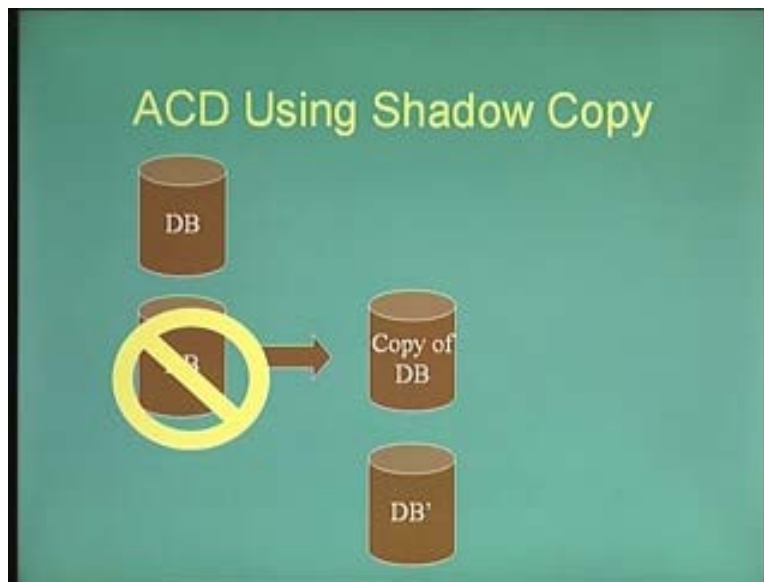
Shadow copy transaction assumes the database to be a single file and assumes that there is only one transaction that is active at any time. Note that it can only provide ACD that is Atomicity Consistency and Durability and not isolation.

(Refer Slide Time: 31:22)



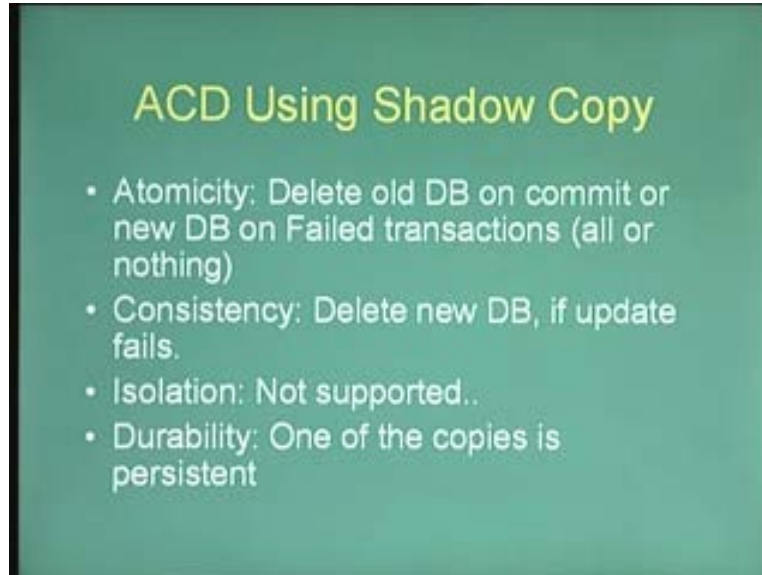
So shadow copy is simply like this. Suppose you have database in a file and you have to perform your transaction. Now before performing your transaction, make a copy of the database that is copy in to entire file. The file and make your changes on the copy of the databases. Now if your changes succeed, that is it does not violate any integrity constraint and it is consistent and it is safe to commit and so on, then simply you delete the original database and then you keep the new updated copy of the database. Incase you have to abort your transaction, then you just delete the copy that you have created and let the original database be in its place as simple as that, that is you make the entire database to in to shadow.

(Refer Slide Time: 32:22)



Copy the entire database into another file and make changes on it and if it is safe to commit the changes, then delete the original file or if it is unsafe delete the new file and let the original file be as it is. Of course, how it is interactional and inefficient but of course later does it satisfy these acid properties of a database. Let us look at atomicity. If i see that i cannot do all operations in a transaction such that atomicity needs to be met then i just delete the new transaction that is i just delete the new file. It is all are nothing.

(Refer Slide Time: 33:10)



When all operations are committed, all operations are performed in the new file, will i delete the old file. So, therefore it is all or nothing. No operations have been performed. Consistency: If any consistency, if any integrity constraint is violated in the new database is deleted. So assign that old database is consistent, we are still left in a consistent state. Isolation obviously not supported because when two or more transactions are copying making different shadow copies cannot we cannot support isolation here and durability. At any point in time, once the transaction commits, it just ensures that either the old file or the new file remains that is once the transaction terminates it is either commit or abort if it commits, then the new file remains. If it aborts, old file remains. So it is durable what are changes made are persistent in the database.

Let us have look at concept of serializability which is again very important, when it comes to recovering from failed transactions. Like we mentioned before, in the previous shadow copy example, isolation was not supported and in order to support isolation we should ensure the notion serializability in our transaction processing environment. This serializability simply says that, if i set of activities from two or more concurrent transaction taking place, they should they should schedule in such a fashion as though transaction were executed in some serial order. So have a look at the slide here. Slide shows two transactions here T1 and T 2 and transaction. Tone is a wire transfer that is taking fifty rupees from A's account putting in to B's account. Transaction T 2 is also a wired transfer that is taking ten percent of whatever amount is their in A's account and crediting in to the B's account.

(Refer Slide Time: 33:25)

Serializability

On executing a set of concurrent transactions on a database the net effect should be as though the transactions were executed in *some* serial order.

Transaction T1: read (A) A = A - 50; write (A) read (B) B = B + 50; write (B)	Transaction T2: read (A) t = A * 0.1; A = A - t; write (A) read (B) B = B + t; write (B)
---	---

Now suppose i have to perform all activities of T1 and then start with all the activities of T 2 obviously it is a serial schedule. Such a schedule is called a serial schedule that is performing all activities of one of transaction before starting first activity of the second transaction. So this equivalent to performing T1 followed by T2.

(Refer Slide Time: 35:27)

Serial Schedules

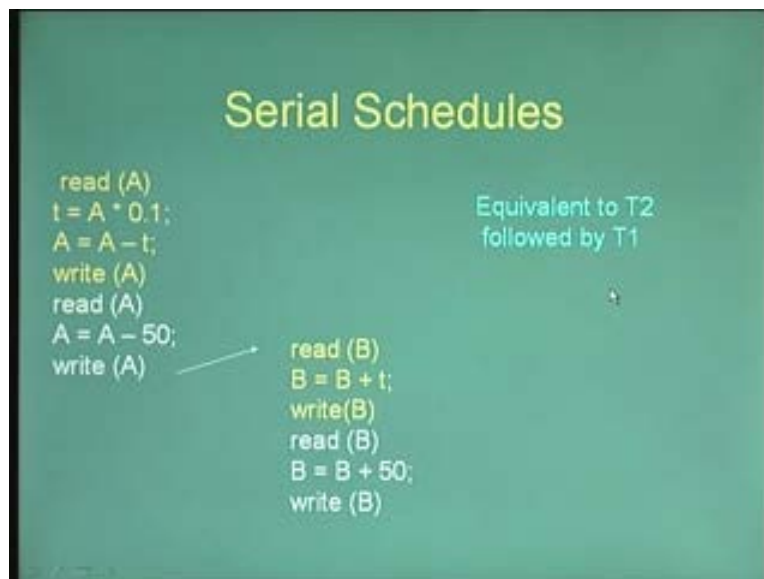
read (A) A = A - 50; write (A) read (B) B = B + 50; write (B)	Equivalent to T1 followed by T2
↗	read (A) t = A * 0.1; A = A - t; write (A) read (B) B = B + t; write (B)

If i perform all activities of T 2 and then start with the first activity of T1 and then perform all activities of T 1, this is also a serial schedule, this is also correct schedule and this is equivalent to T 2 followed by T1. However serial schedules do not have does not necessarily mean that all activities pertaining to given to transaction completed before the first activity of the next transaction T1 is taken up.

For example: this one this slide shows how activities from T1 and T 2 are interleaved. The color activities here belong to T 2 that is read a T equal to eight times “point one” A equal to A minus t write A and then the transaction T2 has not yet completed. But transaction T1 is already begin. Read ‘A’ equal to A minus 50 and so on and then transaction T2 continues here and transaction T1 also continues here. However, if you notice even this schedule is a serialize schedule or it is a serial schedule. This schedule is equivalent to performing T 2 followed by T1.

Why is this so? Have a look here. Have a look at how the activities of T 2 and T1 are interleaved? All activities performing are regarding updation of data element A is completed of the transaction T2 or from the transaction T2 before first operation involved involving data element A is even performed from transaction T1. Same thing with respect to B and we can actually see that, we can rewrite it that is we can take these elements of B back here and put this back here without changing the semantics that is without changing the overall semantics of this serialized schedule that is once this schedule finishes, it is equivalent to as though T2 was executed first followed by T1 that is although activities of T2 before the first activities of T1 ever started.

(Refer Slide Time: 38:15)

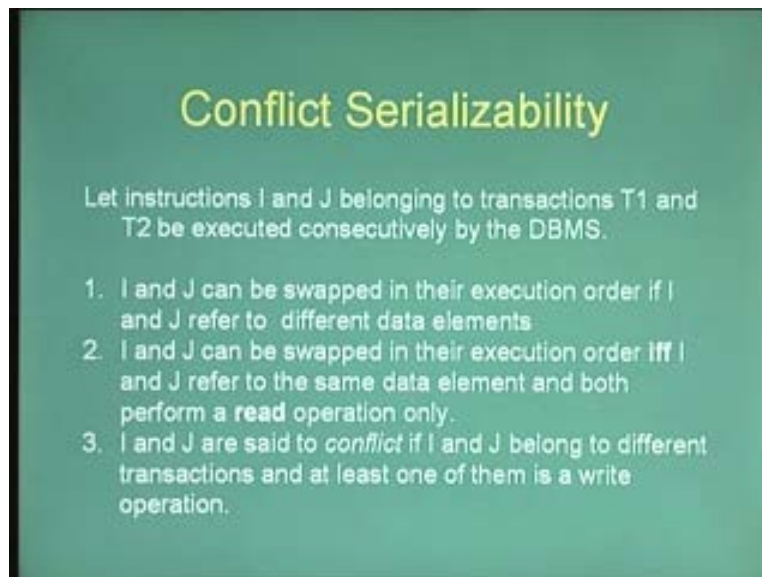


This brings to some definitions of how we can enforce serializability over a set of database activities. We define the term conflict serializability by first defining. The term called conflict between database activities then we say that then we say that particular schedule is conflict serial able. If there is no conflicts or with respect to or when it is

being transformed to a serialize schedule. A schedule in which, all transactions all activities of one transaction is performed before all activities of the second transaction. Now consider 2 activities i and g belong to two different transactions is T1 and T2.

Now i and j can be swapped in their execution order if they refer to different data element because it does not matter. One is referring to element A other is referring to B. It does not matter. We can perform them in any order and i and j can be swapped in their execution order even if they refer to same data element, however all that they are doing is reading all the contents of the data element. Even both are reading the same data element does not matter who is reading first and who is reading second. As long as nothing else in between them that is and they are said to conflict that, i and j cannot be swapped in their execution order if atleast one of them is a write operation. If at least i is trying to write and j is trying to read. We cannot perform i should be read before j, we cannot have j read the database before i writes it and so on.

(Refer Slide Time: 40:04)



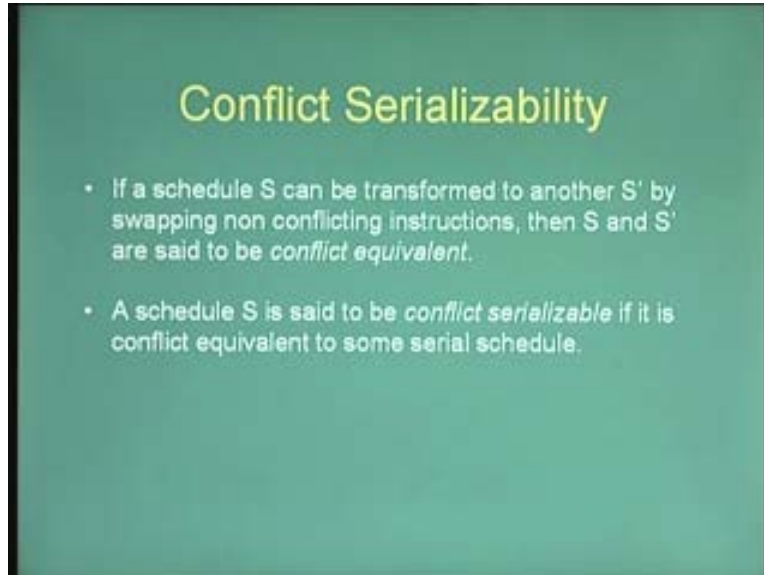
Conflict Serializability

Let instructions I and J belonging to transactions T1 and T2 be executed consecutively by the DBMS.

1. I and J can be swapped in their execution order if I and J refer to different data elements
2. I and J can be swapped in their execution order iff I and J refer to the same data element and both perform a read operation only.
3. I and J are said to *conflict* if I and J belong to different transactions and at least one of them is a write operation.

The same thing is true when both are write operations. Given a schedule 'S'. A schedule is something like what we saw in this slide here, it is a schedule. Now suppose we are given a schedule like this that is the activities of T2 will perform will like this and activities of T1 are performed and activities of T2 continues and so on. Now given a schedule: If we in order to determine whether it is safe or not, whether it is serializable or not, we can identify this, if we can swap or if we make one or more swapping of activities database schedule activities and bring them to a serialize schedule were all activities of one transaction are performed before all activities of the second transactions without encountering any conflicts as a way defined in the previous slide.

(Refer Slide Time: 40:12)

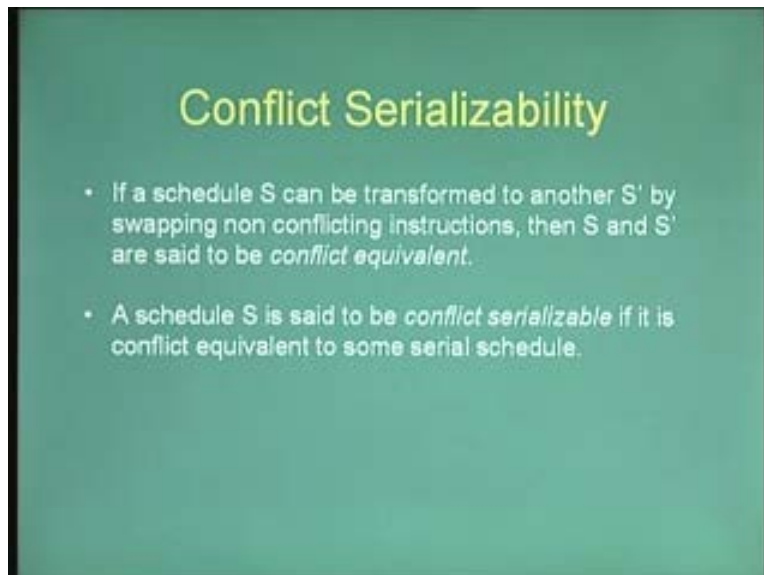


Conflict Serializability

- If a schedule S can be transformed to another S' by swapping non conflicting instructions, then S and S' are said to be *conflict equivalent*.
- A schedule S is said to be *conflict serializable* if it is conflict equivalent to some serial schedule.

Then this kind of schedule is said to be a conflict equivalent schedule and it is also said to be a conflict serializable schedule that is, it can be serialized or it can be equivalent to serialized schedule where the equivalent criteria is conflict equivalence that is conflict serializable.

(Refer Slide Time 41.29)

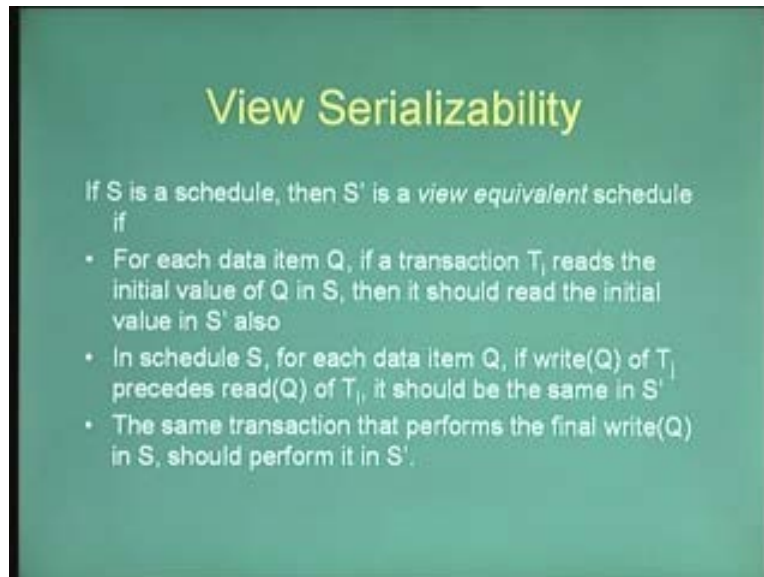


Conflict Serializability

- If a schedule S can be transformed to another S' by swapping non conflicting instructions, then S and S' are said to be *conflict equivalent*.
- A schedule S is said to be *conflict serializable* if it is conflict equivalent to some serial schedule.

There is an alternate weaker notion of serializability called view serializability. Conflict serializability is quite strong and in many cases we do not need the stringent property of conflict serializability. The view serializability simply says the following. Suppose for each data item Q , suppose there are set of transactions that are happening in a dbms system. Now for each data element Q , suppose it was transaction ' T_i ' which reads the initial value of Q in a serialized schedule that is in a serialized schedule ' S ' that is in any other schedule which is view serializable.

(Refer Slide Time: 42:20)



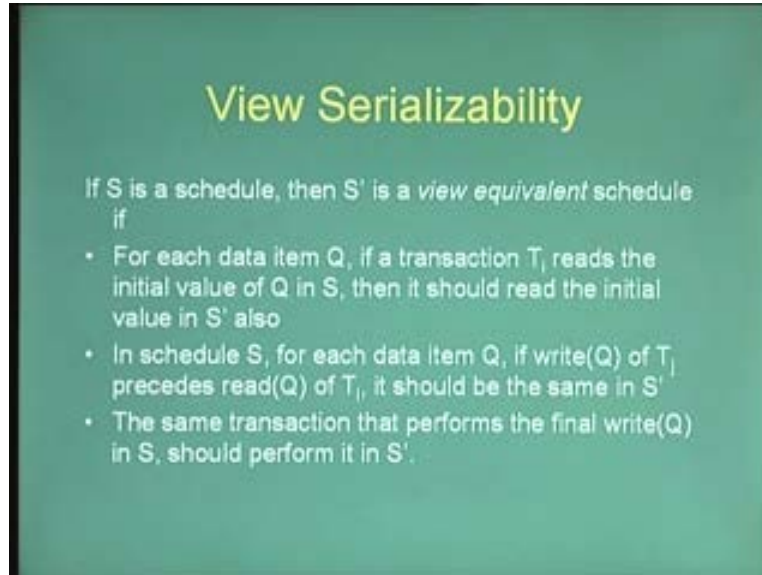
View Serializability

If S is a schedule, then S' is a *view equivalent* schedule if

- For each data item Q , if a transaction T_i reads the initial value of Q in S , then it should read the initial value in S' also
- In schedule S , for each data item Q , if $\text{write}(Q)$ of T_j precedes $\text{read}(Q)$ of T_i , it should be the same in S'
- The same transaction that performs the final $\text{write}(Q)$ in S , should perform it in S' .

It should also be the case that, the same transaction is the first transaction to be reading this data element S or this data element Q and similarly in the given schedule, for each data item Q if T_j precedes or T_j writes to Q before T_i that is before T_i reads, then the same dependency should be maintained in any other schedule that is anybody writing to a data element before somebody else is reading it, this kind of dependency should be maintained in whatever if the schedule has to be view serializable. Similarly, the last operation that is whoever performing the final write in a serializable schedule should be the same transaction who performs the final write in whichever schedule is view serializable.

(Refer Slide Time: 42:38)



Let us take an example: there are 3 transactions shown in this figure. The first transaction T1 reads data element queue and writes it back to disk after performing some operations which is not relevant here. It is only the reads and writes which we have concerned about it and T 2 just writes some something in to queue and T 3 alone it is writes something into Q. Now the following schedule is a view equivalent schedule or a view serializable schedule. Note that it is not a pure serialize schedule. The activities of T1 and T2 are being interleaved here. All activities of T1 are not completed before activities are performed.

However, if you see who is reading the first, who is the first transaction to read the data element Q. Suppose we take a serializable schedule that is T1 is followed by T 2 that is T 2 followed by T3. If in that serialize schedule, the first transaction to be reading data element Q is T1, that is the same thing in the schedule as well. Now is there any read before write dependency? For example: T1 is reading before T2 is writing or rather T1 is reading before T3 is writing. Is that dependency maintained here? that is read before writes write before writes either of those dependencies are maintained here and who is the last transaction writing to Q that is T3 that is same thing here that is T3 is the last transaction that is writing to Q. Therefore this is the view equivalent schedule that is it corresponds to T1 happening before T2 happening before T3, because the first data element to read was T1 read Q was read one last element write in to Q was T3.

So when the T 3 finishes, then there is no difference between saying it was performed as T1 T 2 T 3 or it performed in this fashion. However note that, this schedule is not conflict serializable. If we try to swap updates here, that is if he try to swap the activities here in order to get serialize schedule, then we encounter a conflict that is take a look at the second and third activities here. Now in order to bring in a serialize schedule, we have to swap the second activities with the third activity. So that T1 comes here and T 2 comes

here. However both of them are writes and we saw that when both of them are write operations on the same data element and belong to two different transactions, then you cannot swap them, they are in conflict. Therefore this schedule is not conflict serializable. However it is view serializable that is, as far as database view is concerned, it remains the same whenever we look at before T1 or after T3. It is being the same.

(Refer Slide Time: 46:34)

The slide is titled "View Serializability" in yellow text on a green background. It lists three transactions: T1: Read(Q), Write(Q); T2: Write(Q); and T3: Write(Q). Below this, it states "A view equivalent schedule:" followed by a sequence of operations: T1: Read(Q), T2: Write(Q), T1: Write(Q), and T3: Write(Q).

So every conflict serializable schedule is also view serializable. However, the converse is not true which was the example we saw in the previous slide. Usually, this thing happens that is usually, we find view serializable schedule that are not conflict serializable whenever what are called as blind write. A blind write is something that we saw in the previous slide here.

That is transaction T 2 T 3 shown in slide contains no read operation. They just write in to the database, without any read operation. So such it is a hall mark of blind writes which bring in schedules that are view serializable, but not conflict serializable.

(Refer Slide Time: 47:21)

The slide is titled "View Serializability" in yellow text on a green background. It lists three transactions: T1 (Read(Q), Write(Q)), T2 (Write(Q)), and T3 (Write(Q)). Below this, it states "A view equivalent schedule:" followed by a list of operations: T1: Read(Q), T2: Write(Q), T1: Write(Q), and T3: Write(Q).

Let us look at the last concept of what are called as recoverable schedule and in order to understand what are the requirements of database recovery? Consider the set of following set of transactions as shown in the slide here. There are two transactions here T8 and T9. T8 reads a data element 'A' does some modifications and writes set and then goes about reading about some other read data element and so on. After it writes here transaction T9 reads the data element A and possibly let us say displace it and it does not perform any writes. So, as you can see this is the conflict equivalent schedule That is i can swap read A with read B which will give me serialize schedule that is T8 followed by T9 and swapping by read B and read A it is it is still conflict equivalent. So, therefore performing read A write A read A of T9 and read B of T8 is conflict equivalent.

(Refer Slide Time 48:27)

Recoverable Schedules

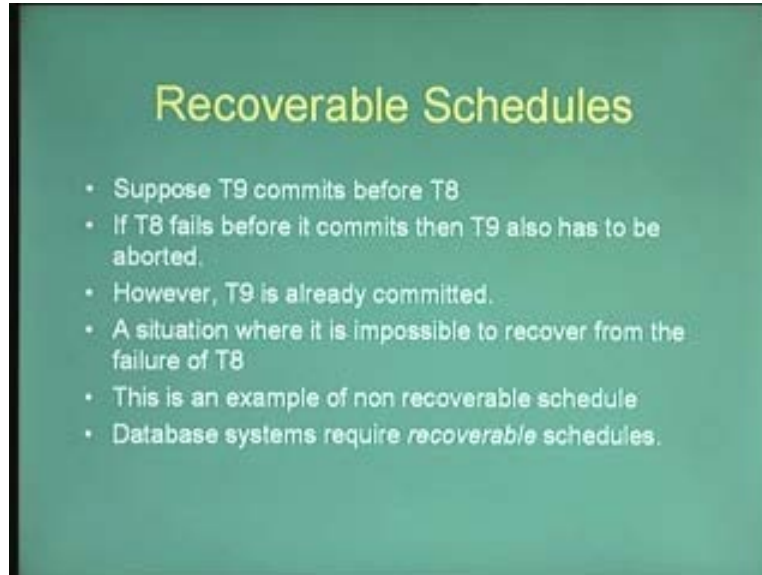
- Consider the following schedule

T8	T9
read(A)	
write(A)	
	read(A)
read(B)	

However, suppose let us say T9 that is read A and displayed that is display the latest value of this stock price or whatever. Suppose this T9 commits and displays the value of the value of k. But T8 is not committed, that is T8 is not completed still and it sees that it cannot commit because some problem somewhere and it has to rollback. Now if it rolls back then T9 also has to be roll backed because it read value of A, after it has been written by T8.

However we cannot rollback T9 because it is already committed and committed is not the durability condition here, that is we have already made some commitment in some sense that is we have displayed the new value of the stock or whatever. So in such a situation, it is impossible to recover from the failure of T8 because we cannot the rolling back of T8 will also require rolling back of T9 it is impossible. So this is in an example of a non-recoverable schedule.

(Refer Slide Time: 49:35)



So serializability or conflict serializability alone is not enough. We need to also look at recoverability of a particular schedule of transaction events if we have to be recover from a database crash. So database system requires a recoverable schedule and finally let us have look at the problem of cascading roll back which is also quite important when it comes to recovery. Even if a schedule is a recoverable, to recover from a failure of a transaction in some times, there is need to roll back several transactions. The previous example was also an example of cascading roll back that is supposes transaction T9 not committed and transaction T8 roll back, then T9 also has to roll back. So in order to make it recoverable we have to defer the commit of T9 until after T8 has committed.

So that will make recoverable. However, it still contains the problem of cascading roll back. So this example also shows cascading rollback situation where there are three transactions T T1 and T 2 and has read a value of A and written it and whatever value is return by T is being read by T1 and T2. Now T1 and T2 cannot commit that is cannot display the new value of A until T has committed. Otherwise, it will become non recoverable. Now even if they do not commit and suppose T has to roll back, it has a cascading effect in T1 and T 2 that is in fact T 2 is dependent on T1 and T1 is dependent on T. So a roll back of T will cause a rollback of T1 which in turn will cause a roll back of T2. So such cascading roll backs will lead to an undoing of large amount of work from several different transactions in case of any database crash or system failure. Cascading rollback is an undesirable thing to happen and leads to an undoing of lot of work.

(Refer Slide Time: 52:05)

Cascading Rollbacks

- Even if a schedule is recoverable, to recover from the failure of a transaction, there is a need to rollback several transactions.

T	T1	T2
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

if T fails, then it will lead to rolling back T1 and T2.
This is an example of cascading rollback.

So when we are looking at schedules of operations that are performed by OLTP environments, they have not to be only serializable. They have to be recoverable cascade list as far as possible we should try to avoid cascading roll backs.

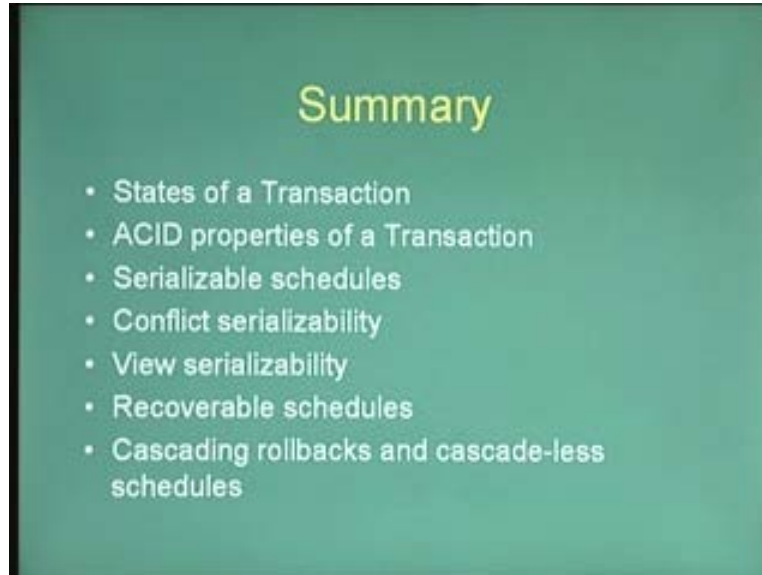
(Refer Slide Time: 52:27)

Cascade-less Schedules

- Cascading rollback is undesirable- leads to undoing a lot of work
- Restrict the schedules to those where cascading rollbacks do not occur.
- Such schedules are cascade-less schedule.

So that brings us to end of the introduction session on database recovery where we have said the ground for all the issues that make up that are concerned or that we have concerned ourselves with whenever we are dealing with database recovery.

(Refer Slide Time: 52:46)



For example: We saw the notion of the transaction that is when we are recovering from the database crash, we have to ensure that we does not leave any transaction in a half or partially committed state. It should either be fully committed or no operation should have performed that is all are nothing. Atomicity transitions have to be obtained whenever we are recovering from database crash or the system crash and in order of that in happen we have recoverable schedules and it is not sufficient for schedules to be serializable and also whether it is conflict or view serializability, it is not sufficient for schedules to be just serializable.

They should also be recoverable schedules and as far as possible they should be cascade list schedule that is crash or rollback of one transaction should not automatically mean that several other transactions or several other work that has been partially completed has to be rollback. It is not even partially completed, even though they are completed just waiting for the commits which what we saw in the operations in the previous slide that is even though transactions T1 and T2 are completed they have read and written a value of a. They are just waiting for the original transaction to commit and because of some problem the original transaction may crash and because of that even all the operations that have been completed have to be rollback without any reason by themselves. So this brings to the end of this session.