**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)

PPD

Week 07 Recap

- Module 31: Transactions/1
  - Transaction Concept
  - Transaction State
  - Concurrent Executions
- Module 32: Transactions/2: Serializabilty
  - Serializability
  - Conflict Serializability
- Module 33: Transactions/3: Recoverability
  - Recoverability and Isolation
  - Transaction Definition in SQL
  - View Serializability
  - Complex Notions of Serializability
- Module 34: Concurrency Control/1
  - Lock-Based Protocols
  - Implementing Locking
- Module 35: Concurrency Control/2
  - Deadlock Handling
  - Timestamp-Based Protocols

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

Database System Concepts - 6th Edition 36.2 ©Silberschatz, Korth and Sudarshan

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)

## Module Objectives

- We need to understand what are the possible sources for failure for transactions in a database
- Various types of storages are used for recovery from failures to ensure Atomicity, Consistency and Durability – these models need to be explored
- To understand recovery scheme based on logging
- To focus on single transactions only

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

Database System Concepts - 6th Edition 36.3 ©Silberschatz

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)

## Module Outline

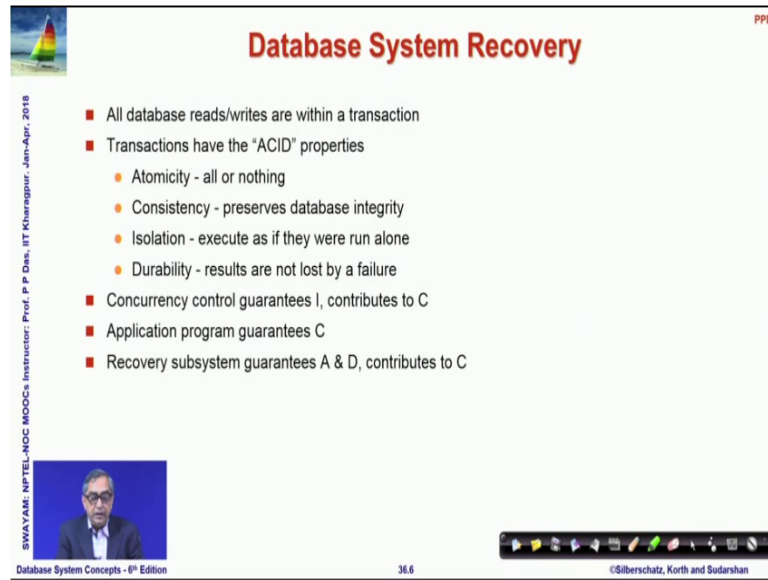
- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

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

Database System Concepts - 6th Edition 36.4 ©Silberschatz, Korth and Sudarshan

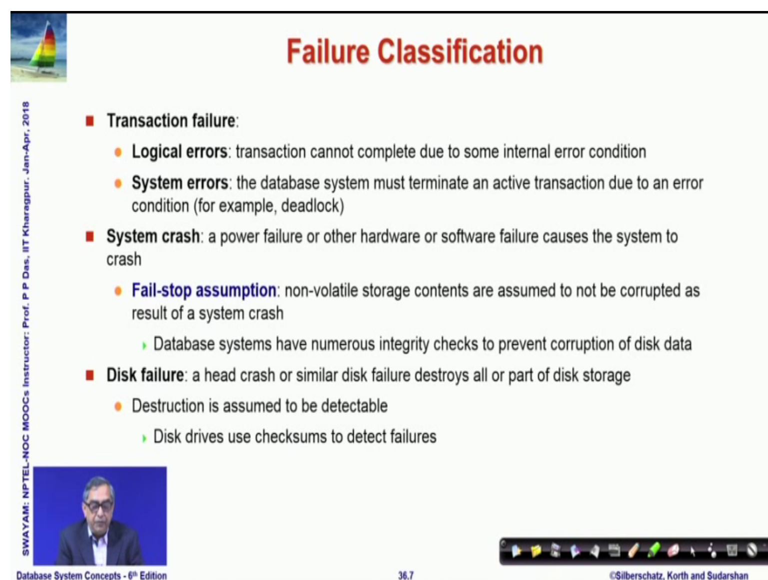
So, these are the topics that will cover.

(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)

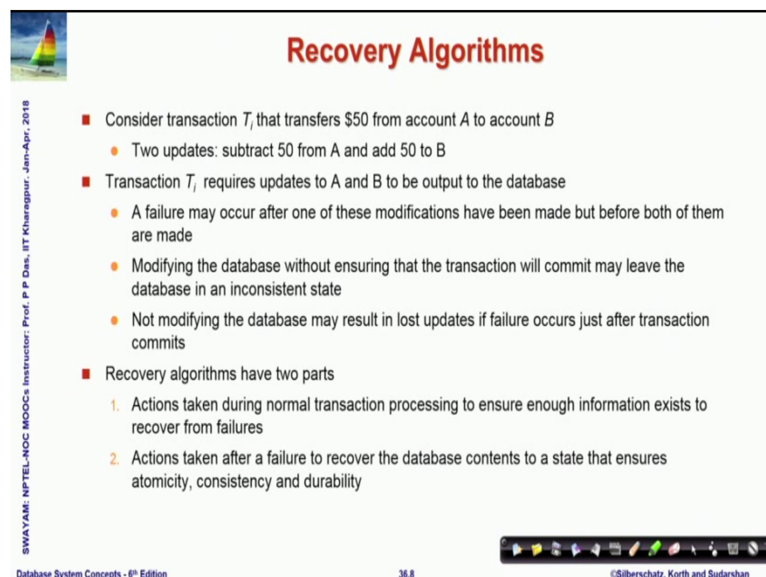


So, let us look at if we are talking about recovery and the phase of failure. So, let us look at what are the generic types of failures that can happen one is the type transactions can fail. A transaction can fail due to logical error due to some internal error or it might fail due to some system error.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



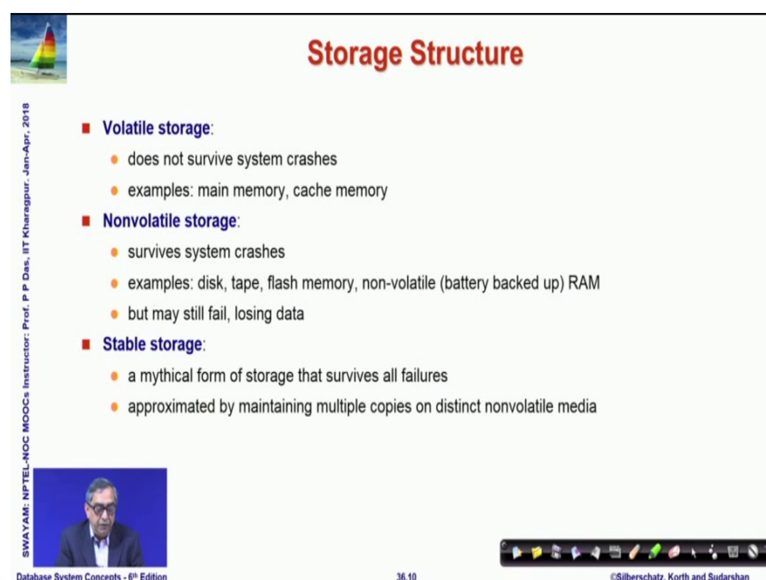

So, in view of that ifs one or more of this failures happen then we need mechanisms to recover from that let us consider a very simple situation of a transaction which we saw earlier to that a transaction T i transfers dollar 50 from account A to account B and

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 informations 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)

Stable-Storage Implementation

- Maintain multiple copies of each block on separate disks
  - copies can be at remote sites to protect against disasters such as fire or flooding
- Failure during data transfer can still result in inconsistent copies. Block transfer can result in
  - Successful completion
  - Partial failure: destination block has incorrect information
  - Total failure: destination block was never updated
- Protecting storage media from failure during data transfer (one solution):
  - Execute output operation as follows (assuming two copies of each block):
    1. Write the information onto the first physical block
    2. When the first write successfully completes, write the same information onto the second physical block
    3. The output is completed only after the second write successfully completes

Secondary storage
Stable database
Read
Write
Local Recovery Manager
Main memory
Fetch, Flush
Database Buffer Manager
Write
Read
Database buffers (Volatile database)

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

Database System Concepts - 6th Edition 36.11 ©Silberschatz, Korth and Sudarshan

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)

## Stable-Storage Implementation (Cont.)

Protecting storage media from failure during data transfer (cont.):

- Copies of a block may differ due to failure during output operation. To recover from failure:
  1. First find inconsistent blocks:
     1. *Expensive solution*: Compare the two copies of every disk block
     2. *Better solution*:
        - Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk)
        - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these
        - Used in hardware RAID systems
  2. If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy
  3. If both have no error, but are different, overwrite the second block by the first block

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

Database System Concepts - 6th Edition 36.12 ©Silberschatz, Korth and Sudarshan

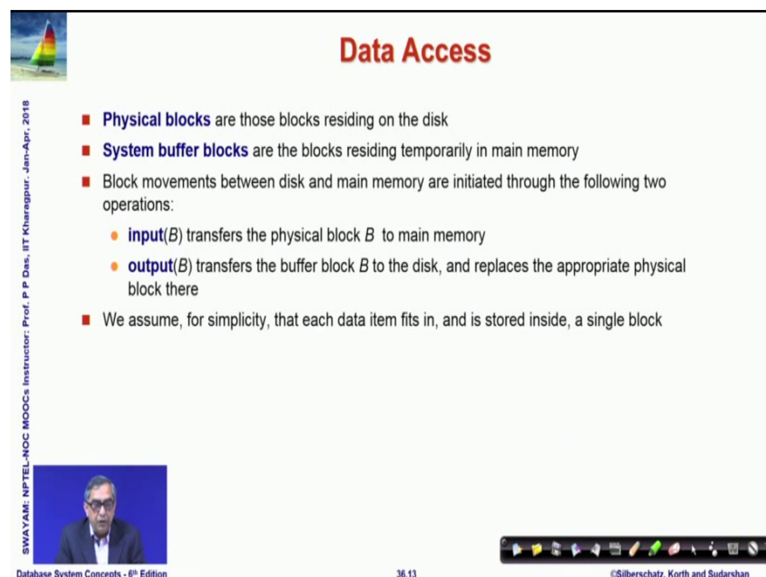
Now, to protect against that ba that happens if during this transfer with during this write if some output operation is some failure happens. So, to recover from that you need to find out what are the blocks which are inconsistent, because we have kept two or more

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)

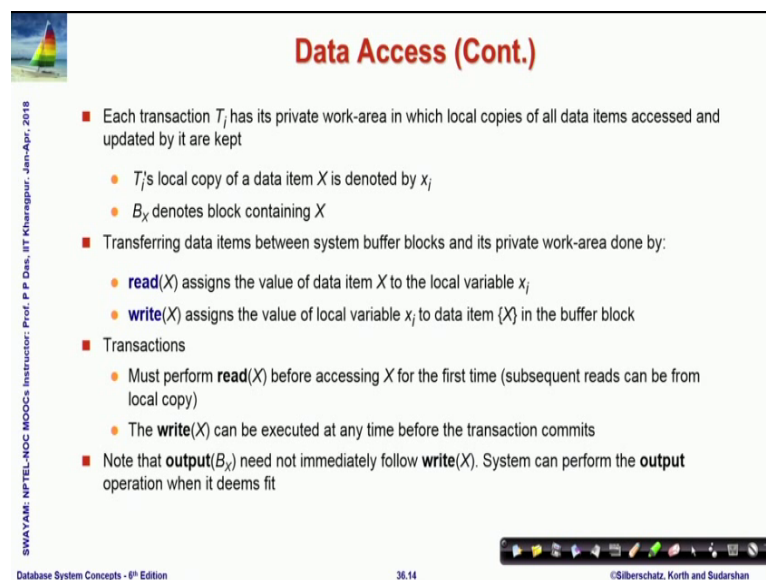



Now, to make this kind of a mechanism work we you resort we have very simple module of data access. We assume that there are physical blocks on the disk that that are on the non-volatile permanent storage and that is where finally, you want your data to decide,

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)



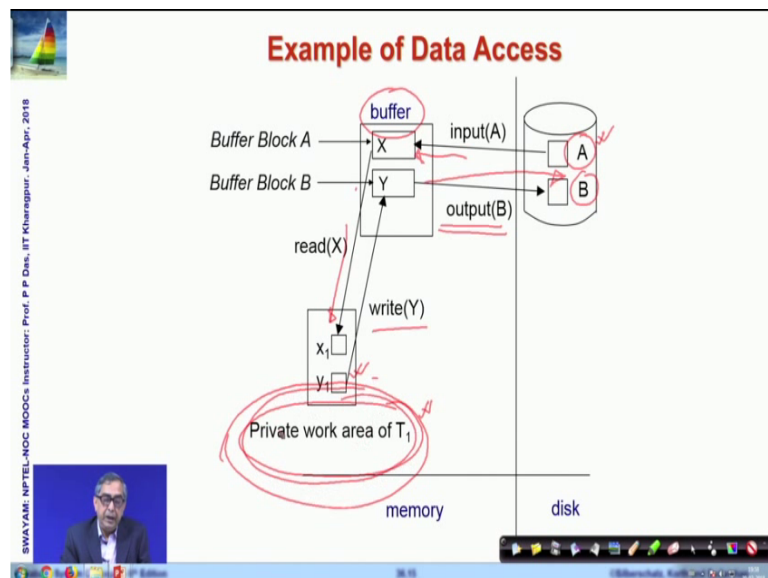

Now, the other part is each transaction on the other side is assumed to have a private work area. So, in the private work area that transaction actually gives local copies and these local copies say you have a data item X, so, you say that for transaction T i the copy of that data item X is xi and say,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

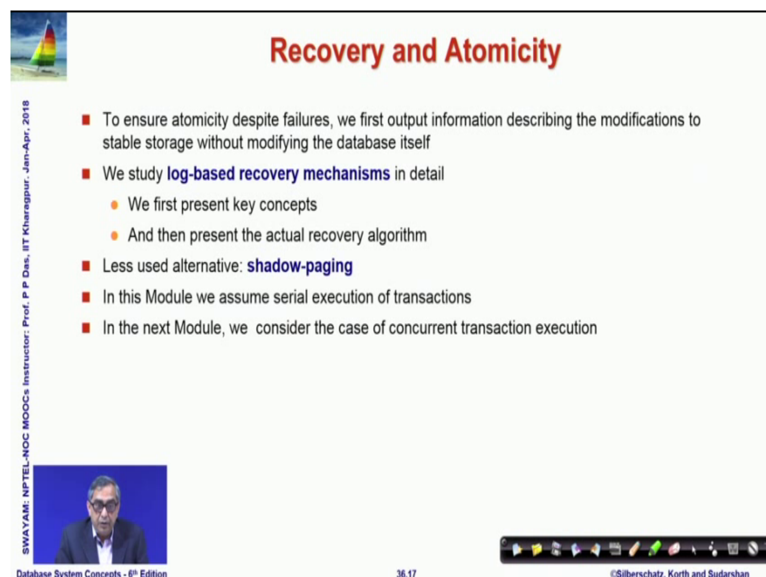
the disk. So, again we will do a write to the buffer area and then at a later point there will be an output which will take this Y back to the disk.

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)

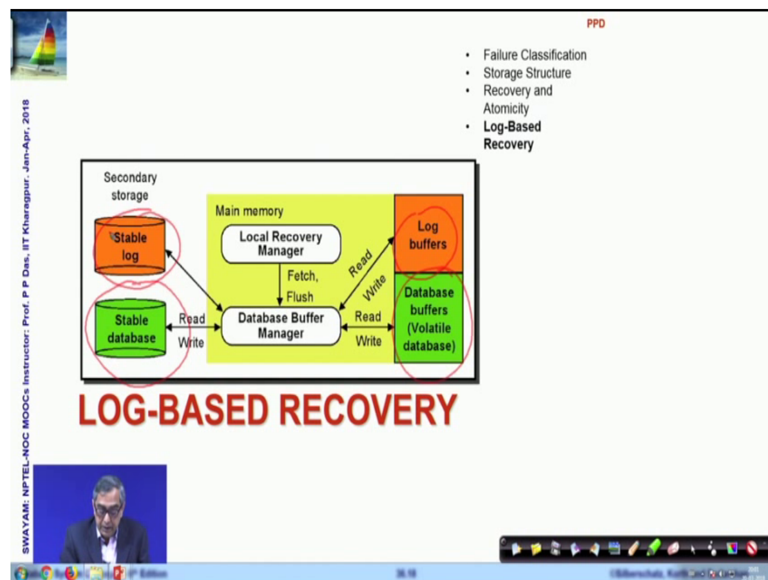


So, to ensure a atomicity in the phase of failure we need to output information describing the modifications to stable storage with input modifying database itself. So, what we are saying that to be able to recover that we should write the changes to the stable storage you recall that stable storage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

## Log-Based Recovery

- A **log** is kept on stable storage
  - The log is a sequence of **log records**, which maintains information about update activities on the database
- When transaction $T_i$ starts, it registers itself by writing a record
  <$T_i$ **start**>
  to the log
- *Before* $T_i$ executes **write**($X$), a log record
  <$T_i$, $X$, $V_1$, $V_2$>
  is written, where $V_1$ is the value of $X$ before the write (the **old value**), and $V_2$ is the value to be written to $X$ (the **new value**)
- When $T_i$ finishes it last statement, the log record <$T_i$ **commit**> is written
- Two approaches using logs
  - Immediate database modification
  - Deferred database modification

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.

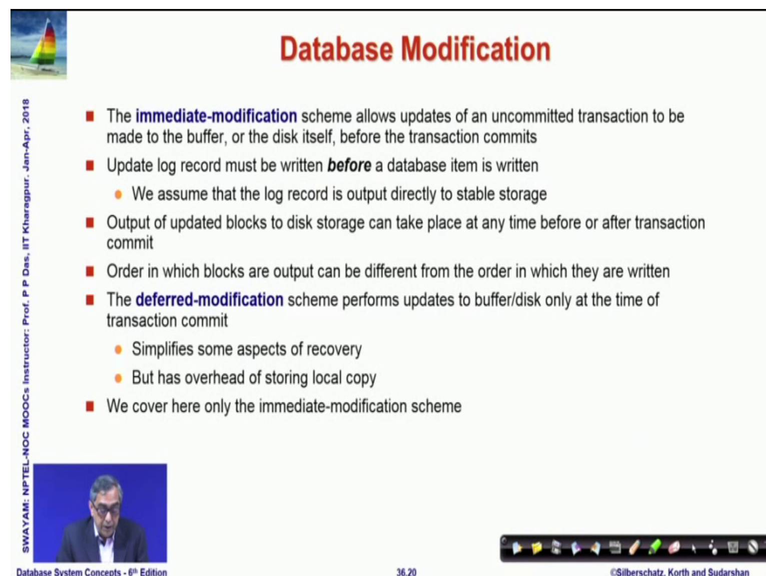
So, here if we look carefully this is the idea of the transaction X is the data item that you want in to write V 1 is the current value of the data item which kind of we can say is the old value and V 2 is the value that we want to actually write. So, here you can see that we are clearly keeping a track 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)



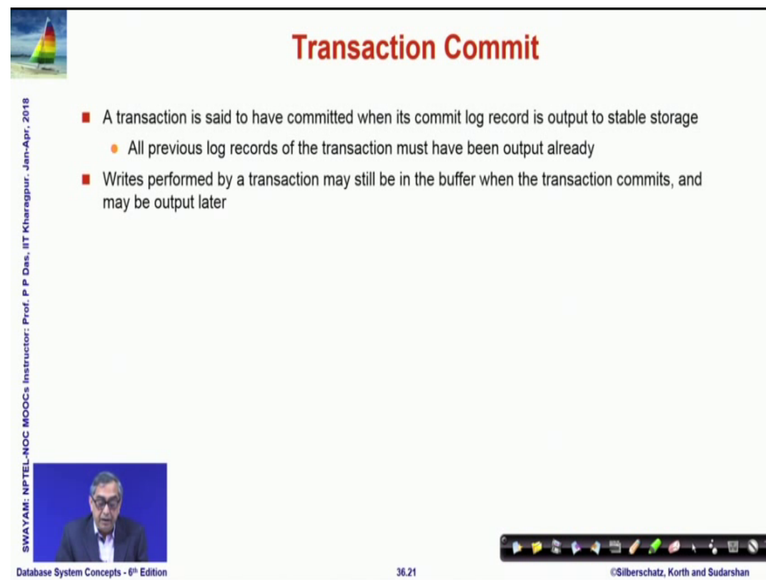

In the immediate modification scheme the ma it allows updates of a uncommitted transaction to be made to the buffer or the disk itself before the transaction commit. So, before the transaction has committed that is before the T i commit log record has been written at that point itself you allow the updates 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.

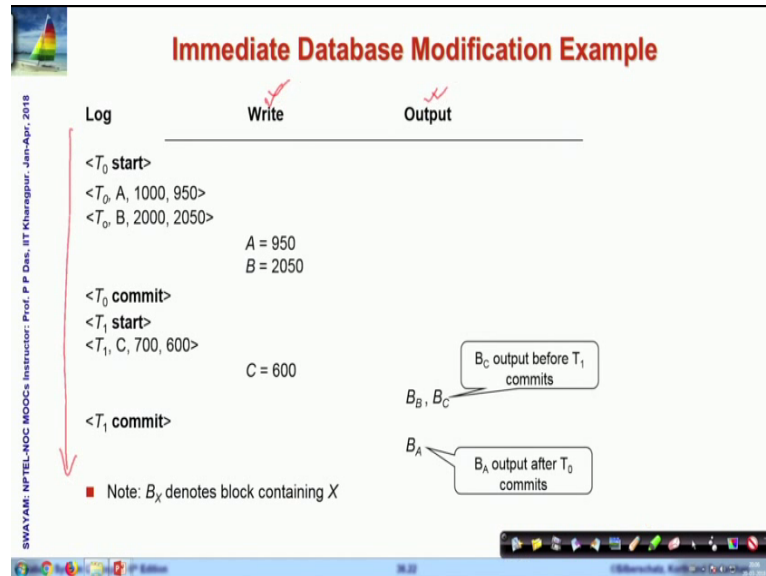
(Refer Slide Time: 22:03)




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 0 has committed whereas, for update of data you can see the output is happening af before the T 1 has committed. So, here it is happening after the commit, but here it is happening before the commit.

So, both of these are permitted both of these are allowed in terms of the protocol that we are following. And, you can also see that in terms of the order in which they were written A was written earlier, but A is output at a later point of time because that is a different sequence in which the system might decide for writing the buffer onto the disk.

So, this is the immediate database modification scheme through which we can write the logs.

(Refer Slide Time: 25:51)

**Undo and Redo Operations**

- **Undo** of a log record $<T_i, X, V_1, V_2>$ writes the **old** value $V_1$ to $X$
- **Redo** of a log record $<T_i, X, V_1, V_2>$ writes the **new** value $V_2$ to $X$
- **Undo and Redo of Transactions**
  - **undo**($T_i$) restores the value of all data items updated by $T_i$ to their old values, going backwards from the last log record for $T_i$
    - Each time a data item X is restored to its old value V a special log record (called **redo-only**) $<T_i, X, V>$ is written out
    - When undo of a transaction is complete, a log record $<T_i$ **abort**$>$ is written out (to indicate that the undo was completed)

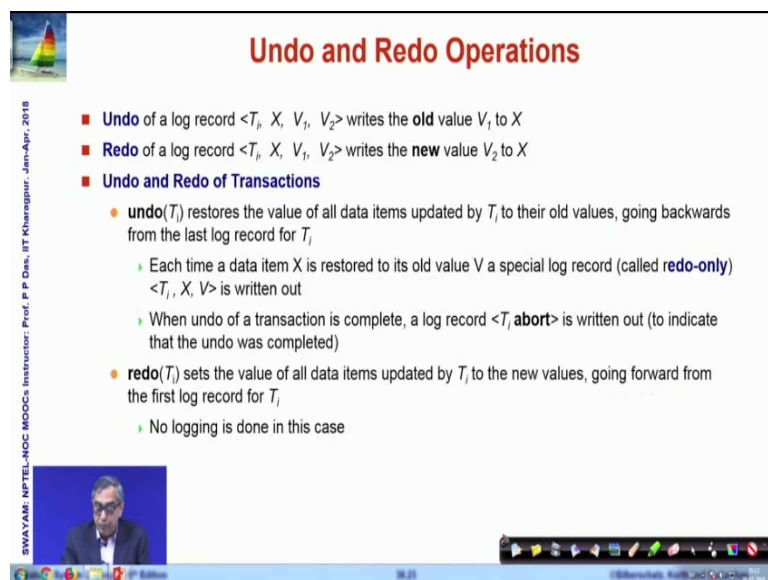
Now, the question is we have written the logs. So, what is the use of those logs? Naturally, the use of those logs are in terms of two operations to which we say are undo operation and redo operation and undo operation is one which basically undoes the operation the effect of an update. So, while ma 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.

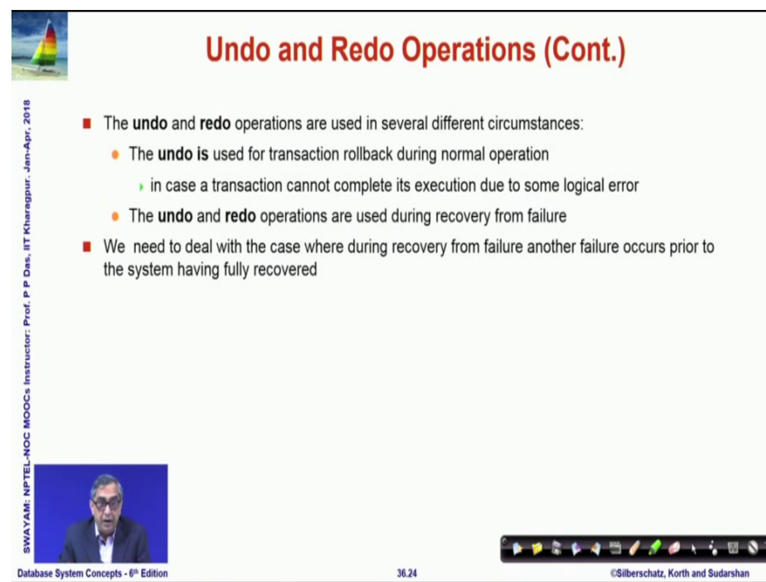
And, every time you restore you write that you write that out as a record which is known as redo, redo only record. So, you can see that here you are not trying to remember the original value you are just writing the value that you have written out in terms of the undo operation that is the old value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

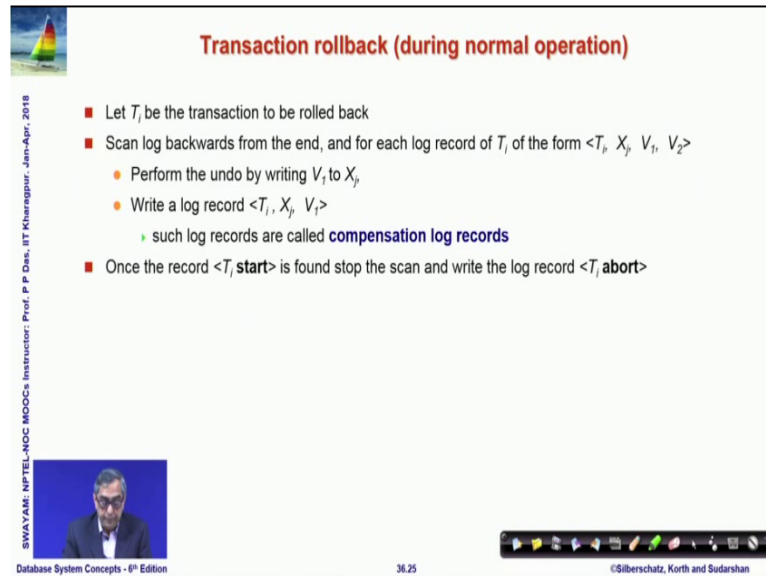
(Refer Slide Time: 29:04)




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 T i start log record when you face come across that you write log record T i 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)

## Undo and Redo on Recovering from Failure

- When recovering after failure:
  - Transaction $T_i$ needs to be undone if the log
    - contains the record <$T_i$ **start**>,
    - but does not contain either the record <$T_i$ **commit**> *or* <$T_i$ **abort**>
  - Transaction $T_i$ needs to be redone if the log
    - contains the records <$T_i$ **start**>
    - and contains the record <$T_i$ **commit**> *or* <$T_i$ **abort**>
  - It may seem strange to redo transaction $T_i$ if the record <$T_i$ **abort**> record is in the log
    - To see why this works, note that if <$T_i$ **abort**> is in the log, so are the redo-only records written by the undo operation. Thus, the end result will be to undo $T_i$'s modifications in this case. This slight redundancy simplifies the recovery algorithm and enables faster overall recovery time
    - such a redo redoes all the original actions including the steps that restored old value – known as **repeating history**

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

Database System Concepts - 6th Edition 36.26 ©Silberschatz, Korth and Sudarshan

In the other case if you are recovering from a failure if there has been a failure then you do something which needs to be understood carefully. So, the transaction T i needs to be undone if the log contains the record start T i start, but it does not contain either T i commit or T i 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 re dones. 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)

**Immediate Modification Recovery Example**

Below we show the log as it appears at three instances of time.

| (a) | (b) | (c) |
|---|---|---|
| <$T_0$ start> | <$T_0$ start> | <$T_0$ start> |
| <$T_0$, $A$, 1000, 950> | <$T_0$, $A$, 1000, 950> | <$T_0$, $A$, 1000, 950> |
| <$T_0$, $B$, 2000, 2050> | <$T_0$, $B$, 2000, 2050> | <$T_0$, $B$, 2000, 2050> |
| | <$T_0$ commit> | <$T_0$ commit> |
| | <$T_1$ start> | <$T_1$ start> |
| | <$T_1$, $C$, 700, 600> | <$T_1$, $C$, 700, 600> |
| | | <$T_1$ commit> |

Recovery actions in each case above are:

(a) undo ($T_0$): B is restored to 2000 and A to 1000, and log records <$T_0$, B, 2000>, <$T_0$, A, 1000>, <$T_0$, **abort**> are written out

(b) redo ($T_0$) and undo ($T_1$): *A* and *B* are set to 950 and 2050 and C is restored to 700. Log records <$T_1$, C, 700>, <$T_1$, **abort**> are written out

(c) redo ($T_0$) and redo ($T_1$): A and B are set to 950 and 2050 respectively. Then *C* is set to 600



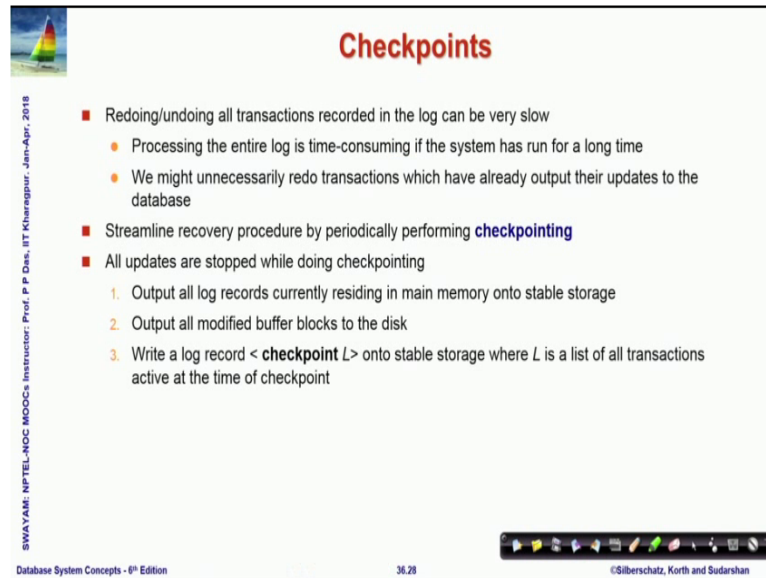
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

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)

## Checkpoints (Cont.)

- During recovery we need to consider only the most recent transaction $T_i$ that started before the checkpoint, and transactions that started after $T_i$
  - Scan backwards from end of log to find the most recent <**checkpoint** *L*> record
  - Only transactions that are in *L* or started after the checkpoint need to be redone or undone
  - Transactions that committed or aborted before the checkpoint already have all their updates output to stable storage
- Some earlier part of the log may be needed for undo operations
  - Continue scanning backwards till a record <$T_i$ **start**> is found for every transaction $T_i$ in *L*
  - Parts of log prior to earliest <$T_i$ **start**> record above are not needed for recovery, and can be erased whenever desired

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

Database System Concepts - 6th Edition 36.29 ©Silberschatz, Korth and Sudarshan

So, during recovery what we need to consider is we have to now we not have to go back to the last time the data 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)

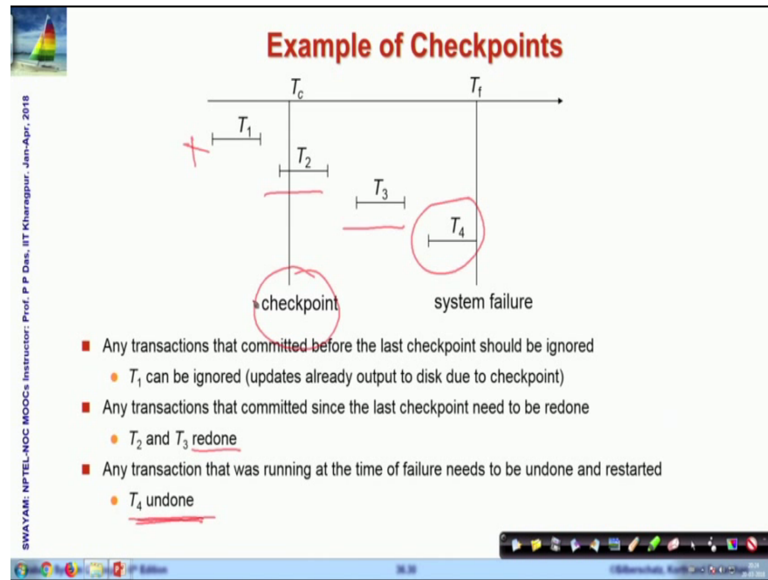
**Example of Checkpoints**

- Any transactions that committed before the last checkpoint should be ignored
  - $T_1$ can be ignored (updates already output to disk due to checkpoint)

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)

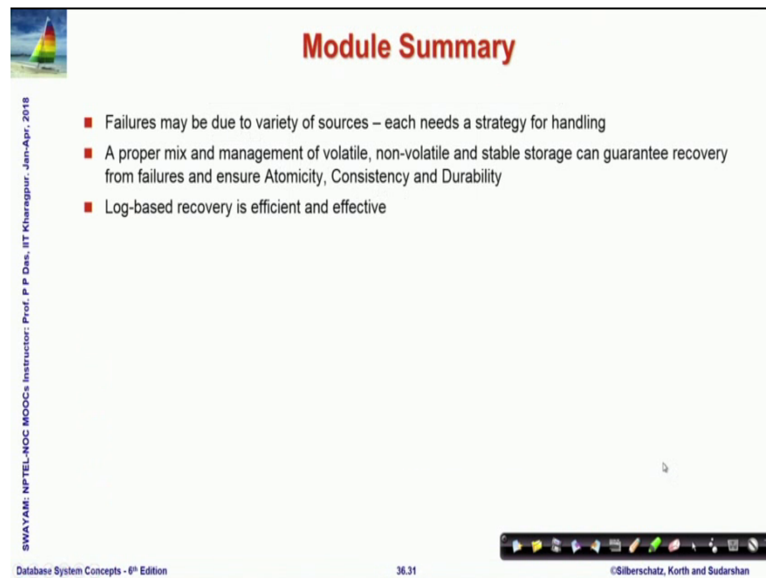



So, if we look at then naturally T 2 if you have a failure at this point if you have a failure at this point as you have here then naturally this is the last stable point you know where everything was written to the disk in a consistent manner. So, what you will need to do you will have to execute transaction 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.