

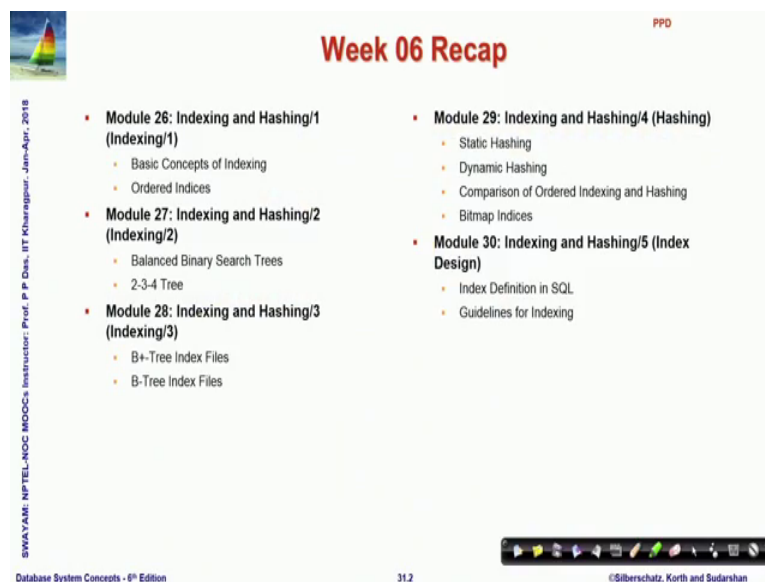
Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 31
Transactions/1

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

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

(Refer Slide Time: 01:40)



PPD

Week 06 Recap

- **Module 26: Indexing and Hashing/1 (Indexing/1)**
 - Basic Concepts of Indexing
 - Ordered Indices
- **Module 27: Indexing and Hashing/2 (Indexing/2)**
 - Balanced Binary Search Trees
 - 2-3-4 Tree
- **Module 28: Indexing and Hashing/3 (Indexing/3)**
 - B+ Tree Index Files
 - B-Tree Index Files
- **Module 29: Indexing and Hashing/4 (Hashing)**
 - Static Hashing
 - Dynamic Hashing
 - Comparison of Ordered Indexing and Hashing
 - Bitmap Indices
- **Module 30: Indexing and Hashing/5 (Index Design)**
 - Index Definition in SQL
 - Guidelines for Indexing

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

Database System Concepts - 6th Edition 31.2 ©Silberschatz, Korth and Sudarshan

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

(Refer Slide Time: 01:46)

PPD

Module Objectives

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

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

Database System Concepts - 6th Edition 31.3 ©Silberschatz, Korth and Sudarshan

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

(Refer Slide Time: 02:03)

PPD

Module Outline

- Transaction Concept
- Transaction State
- Concurrent Executions

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

Database System Concepts - 6th Edition 31.4 ©Silberschatz, Korth and Sudarshan

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

(Refer Slide Time: 02:13)

Transaction Concept

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

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

Database System Concepts - 6th Edition 31.6 ©Silberschatz, Korth and Sudarshan

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

So, it greets possibly makes some local changes and then it writes it back. So, here is an example of a transaction. So, without that detail unnecessary details what it looks at there are 2 accounts account A and B, and we want to transfer 50 dollars from account A to account B. So, we want to we need to debit account A by 50 dollars, and then credit account B by 50 dollars that will achieve the transfer. So, to start this process we first need to know what is the current balance of the account A.

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

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

becomes 300 and 50, and then we write B onto the account based balance. So, account-based balance will now change to 300 and 50 dollar.

So, this whole sequence of 6 instructions is called a transaction. And as you can understand that to achieve our target of transferring 50 dollars from account A to account B, all these 6 instructions have to execute in this order so that we can get the desired results. Now the question is; so, this is pretty simple, this is like a very simple low-level program. But there are 2 main issues that we have to deal with. First, what is the guarantee that once the instruction one starts? What is the guarantee that it will continue up to instruction 6?

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

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

So, they have to occur in a concurrent manner, that is what we see is a concurrent manner because they occur together. And while this is going on, how do we ensure, but there is one CPU. So, the CPU is executing these instructions in some order. So, how do we ensure that this in the face of such concurrency the transactions will still give me correct result? So, these are the 2 major issues for which we are going to study about transactions, and what we in general say the transaction management systems.

(Refer Slide Time: 07:25)

Required Properties of a Transaction

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

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

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

SWAYAM: NPTEL-NOC MBOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 8th Edition

31.7

©Silberschatz, Korth and Sudarshan

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

Suppose again just look at the same transaction, suppose the system crashes there is a system failure after the first 3 instructions has happened and 4th instruction was about to happen. So, what will happen? The already the account A has been debited by 50 dollars and account B has not yet been credited with that 50 dollars. So, simply at this point if the transaction if the system failure happens, then simply 50 dollars will disappear from the system it will not exist. So, the basic requirement is that once a transaction start it should either completely happen it should either do all the 6 instruction as in this case or it should do nothing.

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

(Refer Slide Time: 08:35)

Required Properties of a Transaction

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

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

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

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 6th Edition 31.8 ©Silberschatz, Korth and Sudarshan

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

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

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

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

(Refer Slide Time: 10:28)

Required Properties of a Transaction (Cont.)

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

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

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

Database System Concepts - 9th Edition 31.9 ©Silberschatz, Korth and Sudarshan

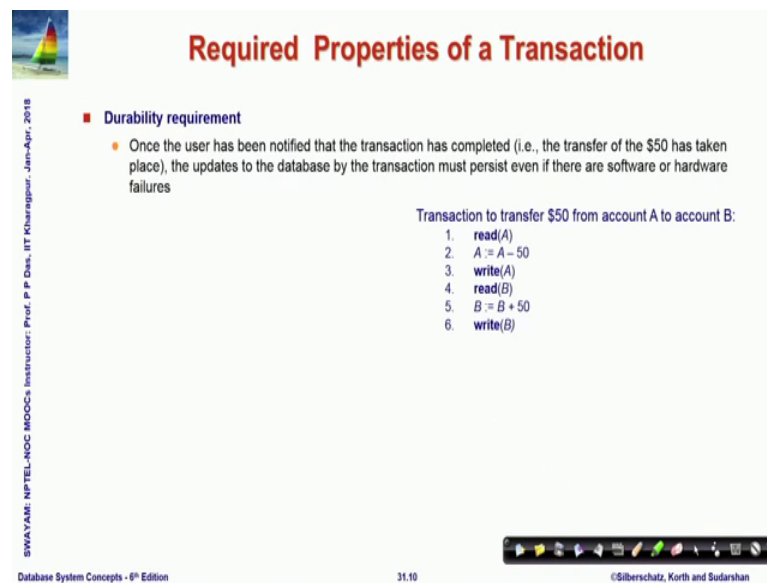
The third is again first look at the example the same on the left is a transaction T 1 which is the transaction we have been talking of. And suppose there is another transaction T 2 which happens concurrently. If it happens concurrently the transaction T 2 has let us say 3 instructions read A read B and print A plus B. So, it tries to read the balance of account A and B and prints their sum.

Obviously, if the transaction T 2 is allowed these 3 instructions of transaction T 2 if they are allowed to be executed; between instruction 3 and instruction 4 of transaction T 1, then T 2 will print a sum of A plus B which is 50 dollars less, than the sum of A plus B at the beginning. So, it will become it will appear as if there is some inconsistency that has happened.

So, the isolation requirement says that when transactions occur concurrently, the net effect of the transactions should be as if they happen either first T 1 happened and then T 2 happen. Over first u or T 2 executed and then T 1 executed. The though even though they can may execute in a concurrent or mixed manner, the result of such inconsistent state of the database should not be available to the other transactions. So, this is called the isolation requirement.

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

(Refer Slide Time: 12:24)



Required Properties of a Transaction

- **Durability requirement**
 - Once the user has been notified that the transaction has completed (i.e., the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures

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

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

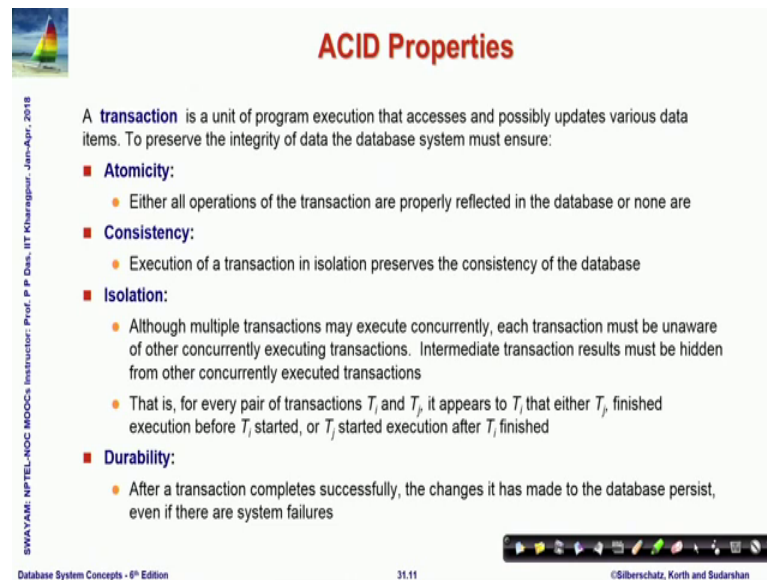
SWAYAM: NPTEL-NOC MCOO+ Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 6th Edition 31.10 ©Silberschatz, Korth and Sudarshan

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

This transaction of transferring 50 dollar from A to B has successfully completed with the 6 instructions having been executed and the money have been transferred, that will should persist even if subsequently some error some failures in the database will occur. So, it must be the changes must be durable.

(Refer Slide Time: 13:11)



The slide is titled "ACID Properties" in red text. It features a small image of a sailboat in the top left corner. The main text defines a transaction and lists four properties: Atomicity, Consistency, Isolation, and Durability, each with a brief explanation and a bulleted list of details. The slide also includes a vertical text on the left side, a footer with the slide number "31.11", and a copyright notice for Silberschatz, Korth and Sudarshan.

ACID Properties

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

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

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

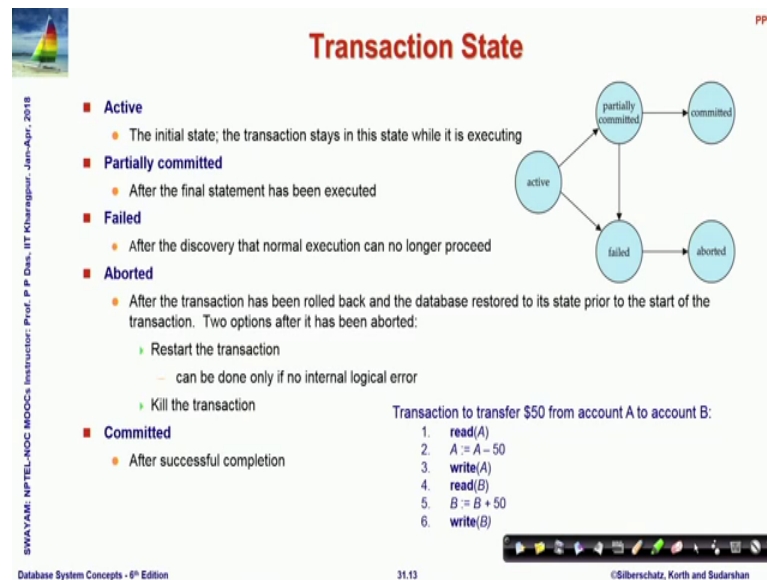
Database System Concepts - 6th Edition 31.11 ©Silberschatz, Korth and Sudarshan

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

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

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

(Refer Slide Time: 14:29)



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

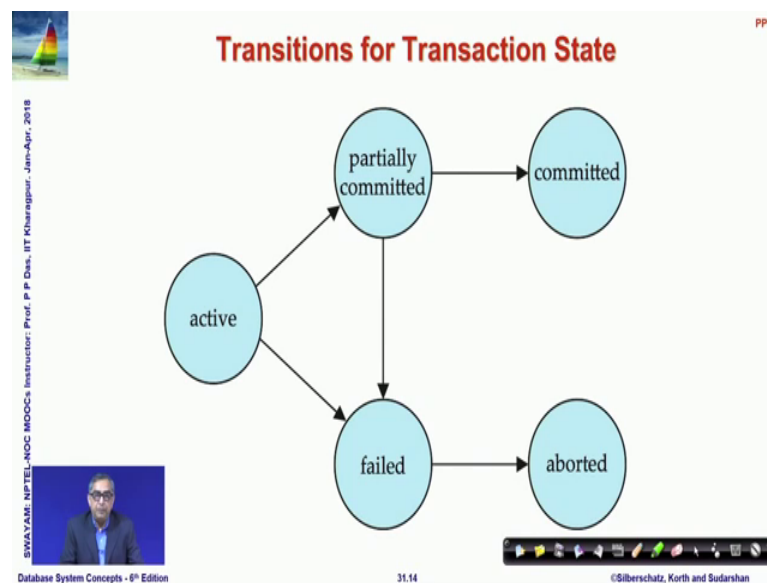
So, once it has executed the last treatment, last instruction that is instruction 6 here, it is in a state that is called partially committed. So, it has been able to successfully complete all the instructions. Or it might happen that during being in the active state, or being in the partially committed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

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

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

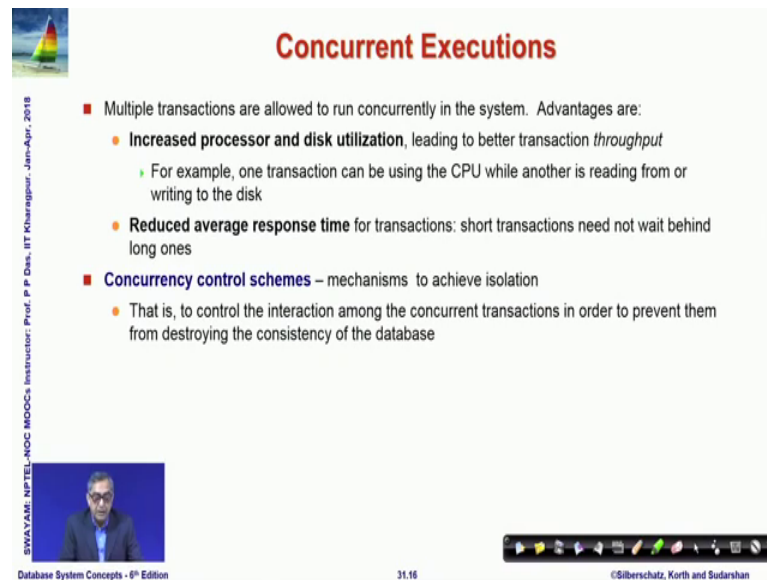
So, every transaction will go through this state at any point of time a transaction will be in one of the states, and depending on the status of execution it will continue to remain in that state or will change state.

(Refer Slide Time: 17:26)



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

(Refer Slide Time: 17:44)



Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
 - **Increased processor and disk utilization**, leading to better transaction *throughput*
 - For example, one transaction can be using the CPU while another is reading from or writing to the disk
 - **Reduced average response time** for transactions: short transactions need not wait behind long ones
- **Concurrency control schemes** – mechanisms to achieve isolation
 - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

SWAYAM: NPTEL-NOC MCOCS Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 9th Edition

31.16

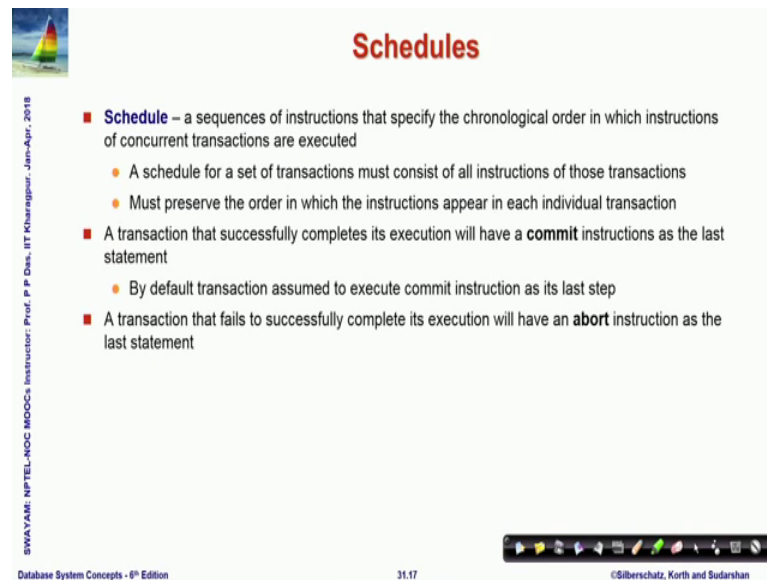
©Silberschatz, Korth and Sudarshan

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

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

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

(Refer Slide Time: 18:50)



Schedules

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

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 6th Edition 31.17 ©Silberschatz, Korth and Sudarshan

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

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

(Refer Slide Time: 20:12)

Schedule 1

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

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

Consistent @ Commit
 Inconsistent @ Transit
 Inconsistent @ Commit

Database System Concepts - 6th Edition 31.18 ©Silberschatz, Korth and Sudarshan

So, let us take an example so, again we are going back to the same example. So, we have 2 transactions T_1 and T_2 . T_1 transfers 50 dollars from A to B as we have seen. And T_2 transfers 10 percent of the balance from A to B. So, one transaction debits 50 dollars one transaction debits 10 percent of the account balance of A to B. So, if they are serially executed as you can see here we are serially executing them as in. So, first you first your whole of T_1 is executing, and once this has committed, that is successfully ended then T_2 is executing.

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

Then it reads B it reads 200 adds that 250 it writes B. You come to this point where after this right, when the commit is happening after the writing this B. Then T_1 has actually completed, and 50 dollars has got transferred to account B, and the sum is again back to 300 so, consistency is preserved. Then transaction 2 starts so, A is red 50 dollars is red in

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

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

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

(Refer Slide Time: 23:05)

Schedule 2

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

T_1	T_2
	read (A) $temp \Rightarrow A * 0.1$ $A \Rightarrow A - temp$ write (A) read (B) $B \Rightarrow B + temp$ write (B) commit
read (A) $A \Rightarrow A - 50$ write (A) read (B) $B \Rightarrow B + 50$ write (B) commit	

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
90	210	300	T2, write B	@ Commit
40	210	250	T1, write A	
40	260	300	T1, write B	@Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

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

Database System Concepts - 9th Edition 31.19 ©Silberschatz, Korth and Sudarshan

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

And you can note that now the end value of a is 40 dollar, and n value of B is 2 60 dollar. In the previous schedule, the value was 45 dollar, and 255 dollar these 2 are different. But both of them are actually correct both of them are consistent, because when things

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

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

(Refer Slide Time: 24:20)

Schedule 3

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

T_1	T_2	T_1	T_2	A	B	A+B	Transaction	Remarks
read (A) $A := A - 50$ write (A)		read (A) $A := A - 50$ write (A)		100	200	300	@ Start	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)	read (B) $B := B + 50$ write (B) commit		50	200	250	T1, write A	
				45	200	245	T2, write A	
				45	250	295	T1, write B	@ Commit
read (B) $B := B + 50$ write (B) commit			read (A) $temp := A * 0.1$ $A := A - temp$ write (A)	45	255	300	T2, write B	@Commit
	read (B) $B := B + temp$ write (B) commit		read (B) $B := B + temp$ write (B) commit					

Schedule 3 Schedule 1

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

Note – In schedules 1, 2 and 3, the sum "A + B" is preserved.

Database System Concepts - 6th Edition 31.20 ©Silberschatz, Korth and Sudarshan

Now, take an interesting example, where schedule 3 where in here if you, if you look at carefully there are few instructions of T 1 which are executed. And then in the temporal order few other instructs, few instructions of T 2 are executed, then again T 1 then again T 2.

So, the instructions from 2 different transactions are getting interleaved. And this is what the execution status would be. So, you can see that when you are when T 1 writes a this is where you are 50 dollars has been debited. Then when T 2 writes a subsequently, another 5 dollar is debited so, it becomes 45. So, then you have T 1 again executing and adding B on to that. And by that it is not only that it has gone into an inconsistent, it is it was already in an inconsistent state, but that was transient that was temporary. But now the transaction T 1 has totally completed. It is completed his execution it is at it is commit, but your database is still in an inconsistent state.

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

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

(Refer Slide Time: 26:58)

Schedule 4

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

T ₁	T ₂
read (A) A := A - 50	read (A) temp := A * 0.1 A := A - temp write (A) read (B)
write (A) read (B) B := B + 50 write (B) commit	B := B + temp write (B) commit

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
90	200	290	T1, write A	
80	250	340	T1, write B	@ Commit
80	280	350	T2, write B	@ Commit

Consistent @ Commit
 Inconsistent @ Transit
 Inconsistent @ Commit

Database System Concepts - 6th Edition 31.21 ©Silberschatz, Korth and Sudarshan

Now, look at again for those transactions look at a different interleaving, a different schedule, again T 1 T 2 are involved. But you have now tried to interleave them in a in a different order. So, earlier the interleaving was done after T 1 has done right, here it is done after he has been the locally debited by 50. And then this part is done and then the

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

Now, if you go through the state you will find that when transaction T 1 commits ends here, you have an inconsistent state. And finally, even when the schedule ends that T 2 has committed. There is A, you are in an inconsistent state somehow that sum of A and B which was 350 has become 300 has become 350 so, 50 dollars as what generated. So, this is so you can see that if you interleave the transactions, then it is quite possible that the transactions will may or may not actually give you a consistent data base.

(Refer Slide Time: 28:07)

Module Summary

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

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

Database System Concepts - 6th Edition 31.22 ©Silberschatz, Korth and Sudarshan

So, here in this module, we have understood the basic tasks that a data bit base performs database executes; which is in form of a transaction. And we have seen that they must satisfy a set of properties typically called the acid properties, and atomicity, consistency, isolation and durability must be satisfied. And when the transactions are executed in concurrent fashion, we improve the throughput, but the concurrent execution of transaction raise issues of serializability; that is, the concurrent execution that the interleaved schedule of instruction of 2 or more transactions can give rise to certain effect which violate the acid properties.

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