

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

Lecture – 34
Concurrency Control /1

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

(Refer Slide Time: 00:28)

PPD

Module Recap

- Recoverability and Isolation
- Transaction Definition in SQL
- View Serializability
- Complex Notions of Serializability

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

Database System Concepts - 6th Edition 34.2 ©Silberschatz, Korth and Sudarshan

So, in the last module we have talked about continuing on the transactions, we had talked about recoverability of databases, how to satisfy the acid properties the basic transaction in SQL and we have introduced a second form of serializability in terms of the view serializability.

(Refer Slide Time: 00:44)

PPD

Module Objectives

- Concurrency Control through design of serializable schedule is difficult in general. Hence we take a look into locking mechanism and Lock-Based Protocols
- We need to understand how locks may be implemented

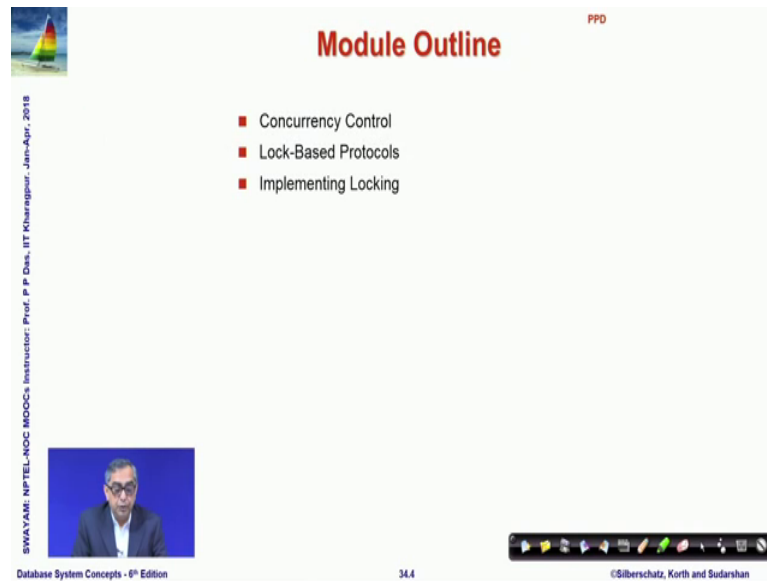
SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 9th Edition 34.3 ©Silberschatz, Korth and Sudarshan

Now, here in this we will talk more on the different aspects of concurrency control because, it is good that if 2 schedules are given, we can we may try to prove if there conflict serializable, if their view serializable and then we can use them, but doing that in general while the database is in execution is an extremely difficult problem because, who is going to give the who is who will be able to give the all possible different types of transactions that may happen 100s of them that may be going on in a in the database at any given point of time.

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

(Refer Slide Time: 01:40)



Module Outline

- Concurrency Control
- Lock-Based Protocols
- Implementing Locking

Database System Concepts - 6th Edition 344 ©Silberschatz, Korth and Sudarshan

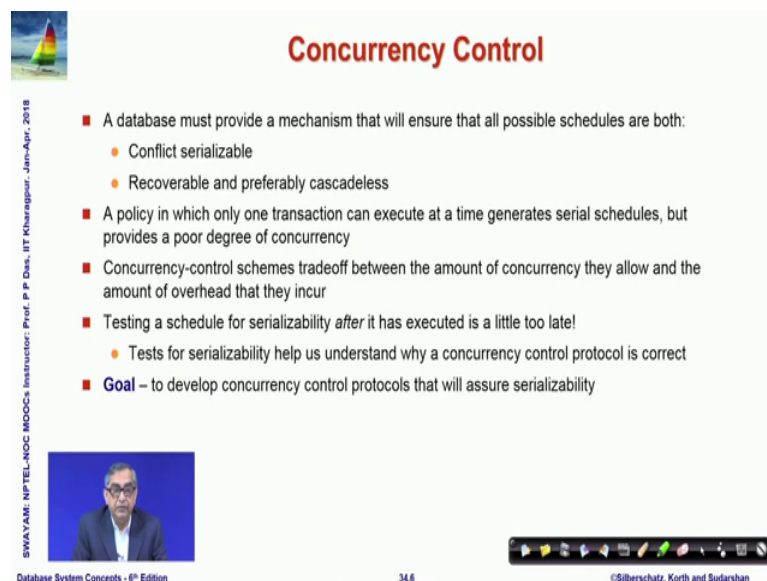
SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

PPD

Navigation icons: back, forward, search, etc.

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

(Refer Slide Time: 01:45)



Concurrency Control

- A database must provide a mechanism that will ensure that all possible schedules are both:
 - Conflict serializable
 - Recoverable and preferably cascadeless
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency
- Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur
- Testing a schedule for serializability *after* it has executed is a little too late!
 - Tests for serializability help us understand why a concurrency control protocol is correct
- **Goal** – to develop concurrency control protocols that will assure serializability

Database System Concepts - 6th Edition 346 ©Silberschatz, Korth and Sudarshan

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

Navigation icons: back, forward, search, etc.

So, a database must provide a mechanism that will ensure all possible schedules are both conflict serializable, that is a basic requirement and are recoverable and preferable in a cascade less manner. So, that is the basic requirements that we have seen.

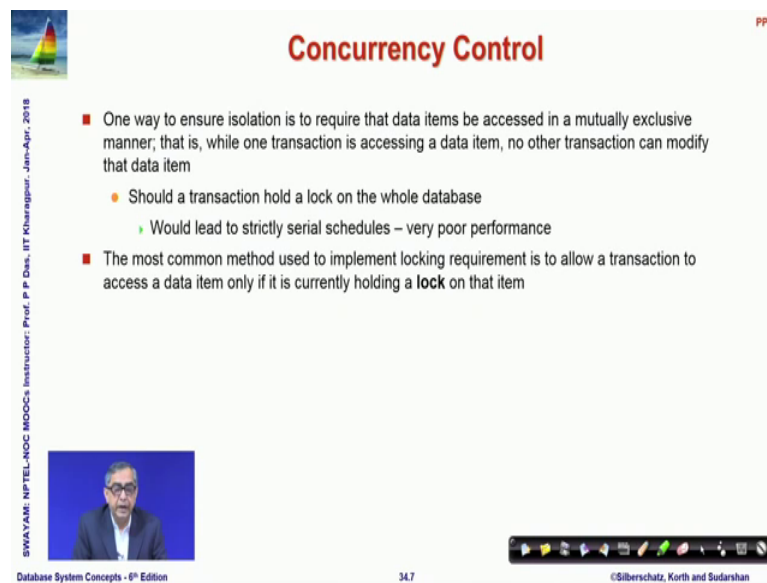
Naturally if we have everything as serial that will happen by default, but that will have very poor degree of concurrency and very low throughput. So, concurrency control

schemes will trade off the amount of concurrency that is allowed and, the amount of overhead. So, what will I have to be ensured is I should be able to, for example, if I say that the schedules are always serial then the overhead of ensuring concurrency is the minimum, but naturally the benefit is also minimum, we get a very poor throughput.

The more we would like to allow for more and more concurrency in the system, but at the same time we will need to have to see what is the overhead of that, what is the cost of that, how do we have to ensure those and, naturally as we have already said testing for a schedule to be serializable after it has happened is; obviously, too late and the question is beforehand how do I get to know it in a general setting.

So, we need to have a certain protocol through which that, transactions might be written, a protocol through which the transactions must operate so, that we can achieve good concurrency in the system. So, here our objective is to develop concurrency control protocols that will ensure serializability and if possible cascadeless recovery.

(Refer Slide Time: 03:28)



The slide is titled "Concurrency Control" in red text. It features a small image of a sailboat in the top left corner. The main content consists of three bullet points:

- One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item
 - Should a transaction hold a lock on the whole database
 - ↳ Would lead to strictly serial schedules – very poor performance
- The most common method used to implement locking requirement is to allow a transaction to access a data item only if it is currently holding a **lock** on that item

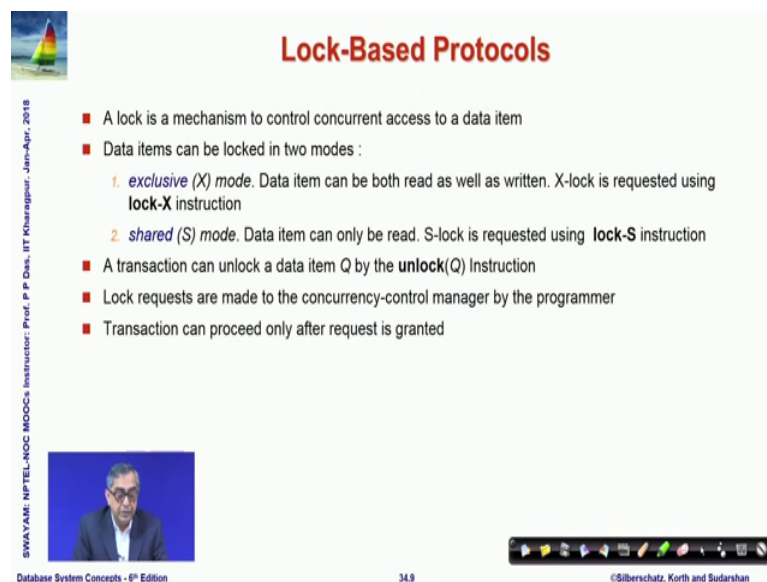
At the bottom left, there is a small video inset showing a man speaking. The slide footer includes the text "Database System Concepts - 6th Edition", the slide number "34.7", and the copyright notice "©Silberschatz, Korth and Sudarshan".

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

So, naturally 1 way it could be to do it using locks. So, we by the basic concept of the lock is you say that this data item is in use. So, others should not use it. Now what should be the data item, should it be the whole database, it can be the whole database we say this database is in use other transaction cannot use it, which boils down to saying almost that you have a serial schedule.

So, at any point of time only one transaction can operate on the database naturally your concurrency will be very poor. So, that is not what is what is acceptable. So, we need locking mechanisms, or a mechanism to control exclusivity in terms of holding locks on smaller items possibly at a record level at a value level and so on. So, that gives rise to a whole lot of lock based protocol some of which we are going to discuss.

(Refer Slide Time: 04:53)



Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
 1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction
 2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction
- A transaction can unlock a data item Q by the **unlock(Q)** Instruction
- Lock requests are made to the concurrency-control manager by the programmer
- Transaction can proceed only after request is granted

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

Database System Concepts - 9th Edition 34.9 ©Silberschatz, Korth and Sudarshan

So, lock is a mechanism to control concurrent access and to start with there are a variety of locks that exist in a database system variety of types, but to start with we are talking about 2 locking modes 1 is exclusive mode, which is designated as X and other is shared mode and which is designated as S.

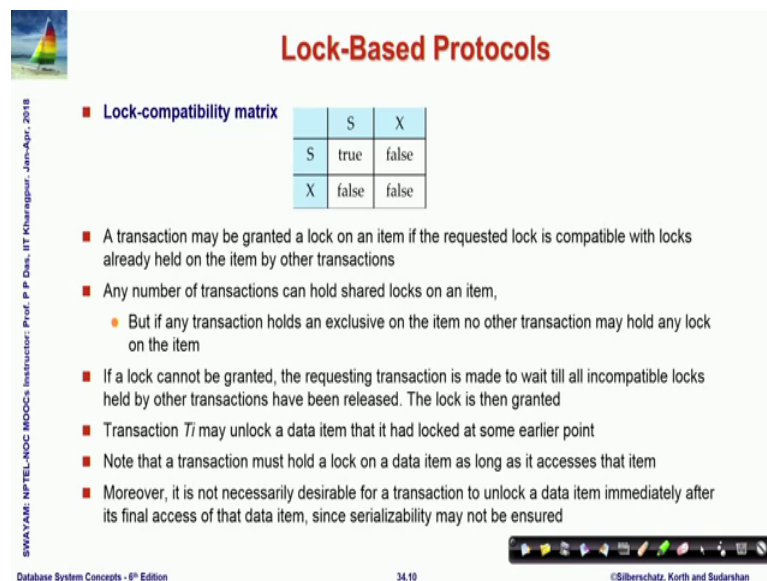
Naturally the exclusive in the exclusive mode, the data item can be read and written both and such a lock is obtained by doing it lock X instruction and in the shared mode the data item can only be read. So, as you can understand why is it exclusive, when you do read write because if 2 transactions try to write the same item at the same time, then you do

not know what is who has been successful and who is the last right and what is the actual final value they will become indeterminate.

But if I have a value and multiple transactions read that at the same time certainly there is no problem because, all of them necessarily will read the same data. So, that is what is called shared and a shared lock or a shared mode lock can be obtained in terms of the lock X instruction. And transaction when it has a lock it can unlock that by an unlock on the same data item.

So, there is a concurrency control manager to whom the lock requests are made, whether it is a request to grant, or it is a request to release and a transaction can proceed only after the request has been granted.

(Refer Slide Time: 06:26)



Lock-Based Protocols

- Lock-compatibility matrix

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
 - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted
- Transaction T_i may unlock a data item that it had locked at some earlier point
- Note that a transaction must hold a lock on a data item as long as it accesses that item
- Moreover, it is not necessarily desirable for a transaction to unlock a data item immediately after its final access of that data item, since serializability may not be ensured

Database System Concepts - 6th Edition 34.10 ©Silberschatz, Korth and Sudarshan

So, let us look into finer details this is what is known as a lock compatibility matrix. So, if there are multiple lock modes which is what is expected, then you try to see which locks can be held or operated simultaneously.

So, this is shown in terms of our present assumption that has shared an exclusive lock naturally, it transact 2 transactions can hold a shared lock simultaneously on the same data item, but all other combinations that is no 2 transactions can hold a shared and an exclusive, or 2 exclusive locks on the same data item at the same time. So, which means that 2 transactions can read a value at the same time, but 2 transactions cannot one is

reading the value and other is writing, the value is not possible the reverse is also not possible and 2 transactions writing, the value is not possible those are called said to be the incompatible modes of loss.

So, if the transaction is granted a lock, if it is compatible with the lock that is already held by another transaction, you cannot get an incompatible lock granted to you. And any number of transactions certainly can hold the shared lock and an item, but if any transaction wants to have an exclusive lock on the item, then no other transaction may hold any lock on that item. So, if I want to write that as a transaction I must be the only transaction who is who has to have that exclusive lock I must be the only transaction who is trying to write, but when I want to read many transactions can simultaneously read.

So, if a lock cannot be granted that if I want a lock either a shared lock, or an exclusive lock and if it cannot be granted, then the transaction has to wait till the all incompatible locks have been released and, only then this lock can be requested lock in being granted. And certainly a transaction who is holding a lock on a data item can unlock it at some point, after it is purpose of accessing the data item is over and, a transaction must hold a lock on the data item as long as it is accessing the item that is the basic protocol.

So, you must request first get a grant of that lock do the operations that you want and then you unlock, this is a basic process that has to happen, usually it is said that as soon as you are done with the operations of the data item you mean unlock that, you may want to wait for a little longer for the ensuring the serializability these details we will see subsequently.

(Refer Slide Time: 09:14)

Lock-Based Protocols: Example

- Let A and B be two accounts that are accessed by transactions T_1 and T_2 .
 - Transaction T_1 transfers \$50 from account B to account A .
 - Transaction T_2 displays the total amount of money in accounts A and B , that is, the sum $A + B$
 - Suppose that the values of accounts A and B are \$100 and \$200, respectively

T_1 :	T_2 :
lock-X(B);	lock-S(A);
read(B);	read(A);
$B := B - 50$;	unlock(A);
write(B);	lock-S(B);
unlock(B);	read(B);
lock-X(A);	unlock(B);
read(A);	display($A + B$)
$A := A + 50$;	
write(A);	
unlock(A);	

- If these transactions are executed serially, either as T_1, T_2 or the order T_2, T_1 , then transaction T_2 will display the value \$300

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

Database System Concepts - 9th Edition 34.11 ©Silberschatz, Korth and Sudarshan

So, let us come to an example. So, here are 2 transactions T_1 and T_2 . So, this is a the 2 transactions T_1, T_2 the transaction T_1 does this operation it transfers 50 dollar from account B to account A .

So, it debits B here credits A here. So, it transfers and transaction T_2 displays the total sum of money in the accounts A and B . So, it reads A reads B displays and say initially the transaction initially let us say these accounts have values 100 and 200. So, what this transaction will do it needs to do the transfer. So, it needs to read B debit and write and what it has to do since it has to read it must have A shared lock. Since it has to write it must have a exclusive lock and, if it has got an exclusive lock it will also be able to read that data. So, what it does it performs an exclusive lock.

So, it requests for an exclusive lock and only on getting that it can do this and, when this is over the purpose is over it unlocks B . Similarly to update a it takes an exclusive lock on a updates and, then releases a lock. Transaction T_2 what it does it has to read and display. So, it does not need an exclusive lock it takes the shared lock reads and unlocks, it again takes a shared lock on B reads and unlocks and finally, displays the 2 data.

Now, if these transactions are executed serially that is T_1 after T_2 or T_2 after T_1 , then the transaction T_2 will always display the value 300 because, 300s is the initial value that we will be able to see if T_2 runs first and T_1 300 is all so, the final value because

only 50 dollar has been transferred from B to A. So, the sum remains same. So, you will be able to see that if T 2 runs after T 1. So, the consistency of the database is maintained.

(Refer Slide Time: 11:32)

Lock-Based Protocols: Example

- If, however, these transactions are executed concurrently, then schedule 1 is possible
- In this case, transaction T2 displays \$250, which is incorrect. The reason for this mistake is that
 - the transaction T1 unlocked data item B too early, as a result of which T2 saw an inconsistent state
- Suppose we delay unlocking till the end

T1: lock-X(B); read(B); B := B - 50; write(B); unlock(B); lock-X(A); read(A); A := A + 50; write(A); unlock(A);

T2: lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A + B)

T ₁	T ₂	concurrency-control manager
lock-X(B)		grant-X(B, T ₁)
read(B)		
B := B - 50		
write(B)		
unlock(B)		
	lock-S(A)	grant-S(A, T ₂)
	read(A)	
	unlock(A)	
	lock-S(B)	grant-S(B, T ₂)
	read(B)	
	unlock(B)	
	display(A + B)	
lock-X(A)		grant-X(A, T ₁)
read(A)		
A := A + 50		
write(A)		
unlock(A)		

Schedule 1

Now, let us see let us consider a schedule written here as schedule 1 and the the transactions are executing concurrently. So, then this is a possible schedule and let us see what will happen. So, what it does this is where the lock exclusive lock on B is held and B is updated, then A is read then B is read display is done and then this update on a has happened. And this is where we are showing that how the grants are happening.


So, as the lock is requested then the request goes to the system that a exclusive lock on B is requested by transaction T 1. So, that subsequently gets granted and only when the grant has happened the corresponding axis can start, but it can be any indeterminate amount of time between the request of the lock which is here and, the actual grant of the lock, but this operation can happen only after the grant has happened.

So, in every case that is what has to be observed right. Now what happens in this schedule what will be the consequence. So, in this schedule if we look at the transaction T 2 will display only 250 dollar, it will not display 300 dollar why because, if you if you look at this carefully, if you look at this carefully this is where B has got updated. So, B has become 50 dollar less. And then the whole of A and B have been read and displayed. So, naturally the total sum is 50 dollar less.

So, even though we have used a lock it has not been able to achieve the required even though, we have used the lock we have not been able to achieve the required serializability. And it is possible to create a schedule where inconsistent data is getting generated T 2 is actually reading an inconsistent data. So, you have seen inconsistent state in terms of here. Why did it happen? This happened because if we look carefully this has happened because, T 1 has unlocked to prematurely T 1 has unlocked as soon as the update to be was over.

So, it was possible for T 2 to read that value of B which is not what is desirable and we will see that we might want to delay the unlocking till the end, let us see what happens if we do that.

(Refer Slide Time: 14:43)



Lock-Based Protocols: Example

- Delaying unlocking till the end, T1 becomes T3 and T2 becomes T4

T3:

```
lock-X(B);
read(B);
B := B - 50;
write(B);
lock-X(A);
read(A);
A := A + 50;
write(A);
unlock(B);
unlock(A)
```

T4:

```
lock-S(A);
read(A);
lock-S(B);
read(B);
display(A + B);
unlock(A);
unlock(B)
```

- Hence, sequence of reads and writes as in Schedule 1 is no longer possible
- T4 will correctly display \$300

	T ₁	T ₂	
	lock-X(B)		
	read(B) B := B - 50 write(B) unlock(B)		grant-X(B, T ₁)
		lock-S(A)	
		read(A) unlock(A) lock-S(B)	grant-S(A, T ₂)
		read(B) unlock(B) display(A + B)	grant-S(B, T ₂)
	lock-X(A)		grant-X(A, T ₁)
	read(A) A := A - 50 write(A) unlock(A)		

Schedule 1

So, we are here now it is the same transaction in terms of the notion, but T 1 has been made to T 3 here, where you have seen that unlocking is been pushed to the end T 2 has been made into T 4, where the unlocking is pushed to the end. And now naturally if you look into this, if you wanted to do a schedule 1 you cannot do this kind of a schedule 1 that schedule will not be permissible because, you will not be able to get the locks, T 4 will not be able to get the locks that T 2 could get in the sequence of reads and writes in schedule 1 is no longer possible.

(Refer Slide Time: 15:52)

Lock-Based Protocols: Example

- Given, T3 and T4, consider Schedule 2 (partial)
- Since T3 is holding an exclusive mode lock on B and T4 is requesting a shared-mode lock on B, T4 is waiting for T3 to unlock B
- Similarly, since T4 is holding a shared-mode lock on A and T3 is requesting an exclusive-mode lock on A, T3 is waiting for T4 to unlock A
- Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution
- This situation is called **deadlock**
- When deadlock occurs, the system must roll back one of the two transactions.
- Once a transaction has been rolled back, the data items that were locked by that transaction are unlocked
- These data items are then available to the other transaction, which can continue with its execution

T3:

```
lock-X(B);
read(B);
B := B - 50;
write(B);
lock-X(A);
read(A);
A := A + 50;
write(A);
unlock(B);
unlock(A)
```

T4:

```
lock-S(A);
read(A);
lock-S(B);
read(B);
display(A + B);
unlock(A);
unlock(B)
```

T ₃	T ₄
lock-X(B)	
read(B)	
B := B - 50	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

©Silberschatz, Korth and Sudarshan

So, whatever way we actually do the schedule T 4 will always correctly show, that the sum is three hundred dollar. So, here we are again showing T 3 T 4 this is a schedule given schedule 2, which is just given partially. And since T 3 is holding an exclusive lock on B and T 4 is requesting a shared lock. So, if I hold it this is T 3 is holding an exclusive lock and T 4 is requesting for a shared lock.

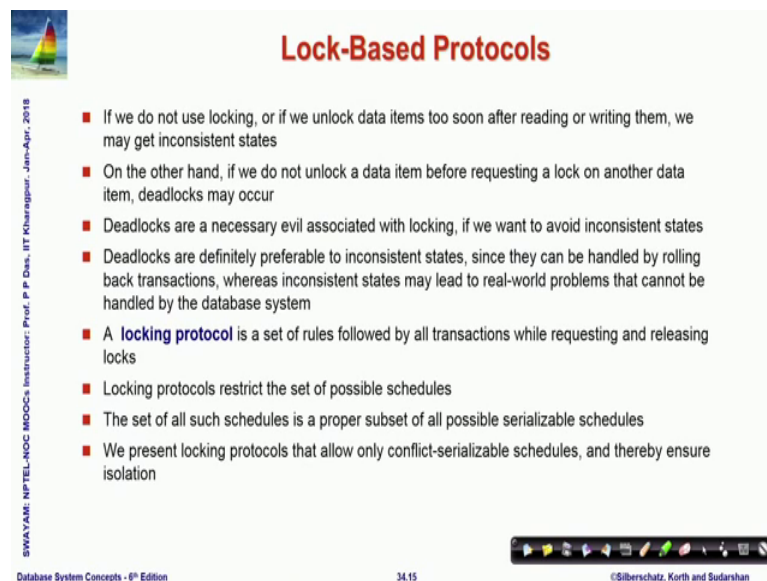
So, T 4 has to wait for T 3 to unlock B before it can actually do that operation. Similarly you will find if you look further T 4 has already got a shared lock to read A. And T 3 needs a shared lock on I am sorry T 3 needs an exclusive lock on A to be able to proceed. So, this 1 is here. So, T 4 cannot go beyond this point because T 3 has the lock on B and, 1 is this T 3 cannot go beyond this point because T 4 has that shared lock.

So, what we situation are we getting into. So, we are getting into a situation where, neither of T 3 or T 4 can actually proceed the normal execution, T 3 is waiting for exclusive lock on A and which T 4 has and T 4 is waiting for the shared lock on B, which T 3 already holds as an exclusive manner. And this in so, this is kind this is what is called deadlock, if you have a studied operating system, then you have must be knowing deadlock very well, and there the deadlock happens 2 different other issues of sharing resources here it is because of the lock.

So, moment you use locks there is a possible danger of having deadlock. And once you have a deadlock there is no other way than to unroll 1 or more of the transaction, then start all over again, it has to roll back 1 of the 2 transactions to be able to proceed. And once the transactions are rolled back the data items that were locked by the transactions will also be unlocked. So, please understand this in view of the earlier discussion we had in terms of transact TCL commands.

So, when you actually which we are roll back we had at that point could only say that your value of the data item in the database will be rolled back, but certainly as now you can understand that, if you roll back also the locks that you have required we also get unlocked so, that other transactions can get those locks and proceed. So, then the data items become available for other transactions and, that can continue the execution.

(Refer Slide Time: 19:03)



Lock-Based Protocols

- If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states
- On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur
- Deadlocks are a necessary evil associated with locking, if we want to avoid inconsistent states
- Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back transactions, whereas inconsistent states may lead to real-world problems that cannot be handled by the database system
- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks
- Locking protocols restrict the set of possible schedules
- The set of all such schedules is a proper subset of all possible serializable schedules
- We present locking protocols that allow only conflict-serializable schedules, and thereby ensure isolation

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

Database System Concepts - 9th Edition 34.15 ©Silberschatz, Korth and Sudarshan

So, if we do not so, so we are saying that we wanted to use locks to get better control on the serializability and so on. And it was partly possible, but then we are getting into different other kinds of different problems.

So, if you do not use locking or, if we unlock data items very early then after reading, or writing them then we may get inconsistent state this is what you have seen, on the other hand, if we do not unlock a data item before requesting a lock on another data item that is if we hold it on for a very long time, then deadlock may occur. So, if we do it too soon

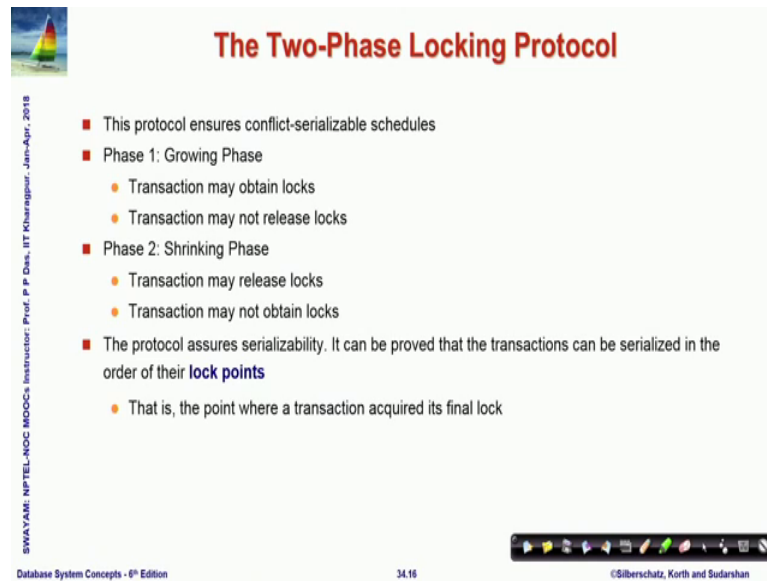
we do not use the lock that we have a problem of inconsistent state, if we do it hold it for too long then there could be problem of deadlock.

Now, deadlocks are necessarily evil of locking, if you do locking you will always face a deadlock, if we want to avoid inconsistent states. Now between these two; obviously, we would prefer deadlock the reason, we will prefer deadlock to inconsistent state is the fact that, if we have deadlock we still have the option of rolling back and we can take different strategies to decide what to rollback and how much to rollback and so on whereas, inconsistent states may lead to real world problems that cannot be handled by the database system.

In fact, in some in many cases I may get into some inconsistent state which is very difficult to even recognize that it is an inconsistent state. So, we will continue and prefer deadlocks over inconsistent states and, we will define we will try to define different locking protocols a set of rules that the transactions should follow, while the request and release locks to make our life relatively easier.

So, locking protocols necessarily will restrict the set of possible schedules because, we will put in some discipline in terms of how we lock and how we release them, and the set of all such schedules is a proper subset of possible serializable schedules that is easy to understand. And we will present locking protocols that allow only conflict serializable schedule which ensures isolation.

(Refer Slide Time: 21:23)



The Two-Phase Locking Protocol

- This protocol ensures conflict-serializable schedules
- Phase 1: Growing Phase
 - Transaction may obtain locks
 - Transaction may not release locks
- Phase 2: Shrinking Phase
 - Transaction may release locks
 - Transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points**
 - That is, the point where a transaction acquired its final lock

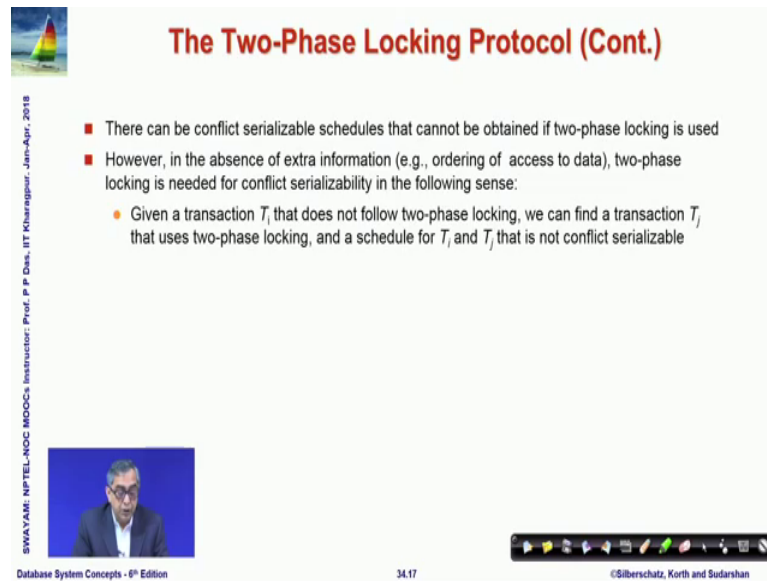
SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 9th Edition 34.16 ©Silberschatz, Korth and Sudarshan

So, let us look at the most widely used protocol this is called the 2 phase locking protocol which guarantees conflict serializability, it does a simple thing it has 2 phases a growing phase, where it transaction may obtain locks and may not release any lock. And a shrinking phase which the transaction may release locks and may not obtain any law. So, you are just separating out the you know the grant or the access of locks holding of locks and the releasing of locks into 2 different phases you do not mix them up.

And that is the 2 phrases phases of the locking protocol. And this ensures so, we are we will not do the proof, but you can look it up in the book or, but you can see through examples that it can be shown that transactions can be serialized in the order of the points where they do the locking. So, these are known as lock points and, that is where the transaction actually acquired it is final block.

(Refer Slide Time: 22:25)



The Two-Phase Locking Protocol (Cont.)

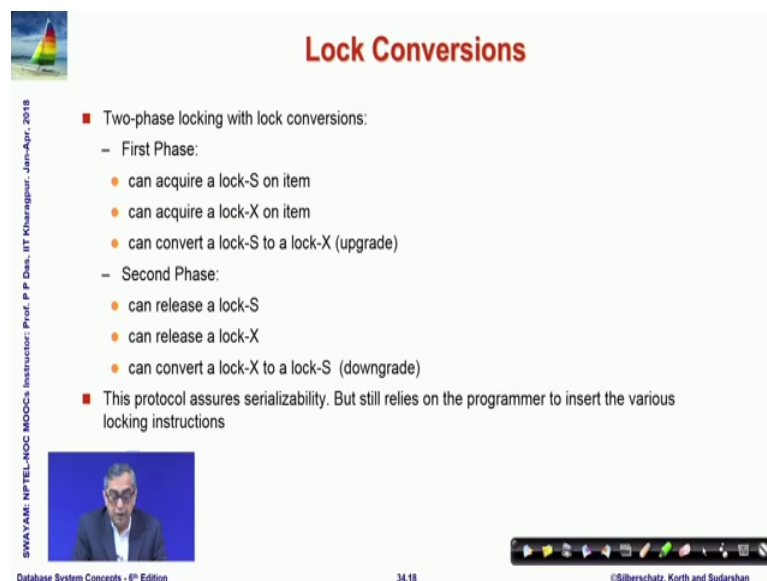
- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
 - Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable

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

Database System Concepts - 6th Edition 34.17 ©Silberschatz, Korth and Sudarshan

So, there can be conflict serializable schedules that cannot be obtained, if 2 phase locking is used. So, what this is saying if you use 2 phase locking you are guaranteed to have conflict serializable schedule, but there are conflicts serializable schedules for which you may not be able to honor the 2 phase locking protocol. So, 2 phase locking protocol is kind of a sufficiency condition.

(Refer Slide Time: 22:50)



Lock Conversions

- Two-phase locking with lock conversions:
 - First Phase:
 - can acquire a lock-S on item
 - can acquire a lock-X on item
 - can convert a lock-S to a lock-X (upgrade)
 - Second Phase:
 - can release a lock-S
 - can release a lock-X
 - can convert a lock-X to a lock-S (downgrade)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions

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

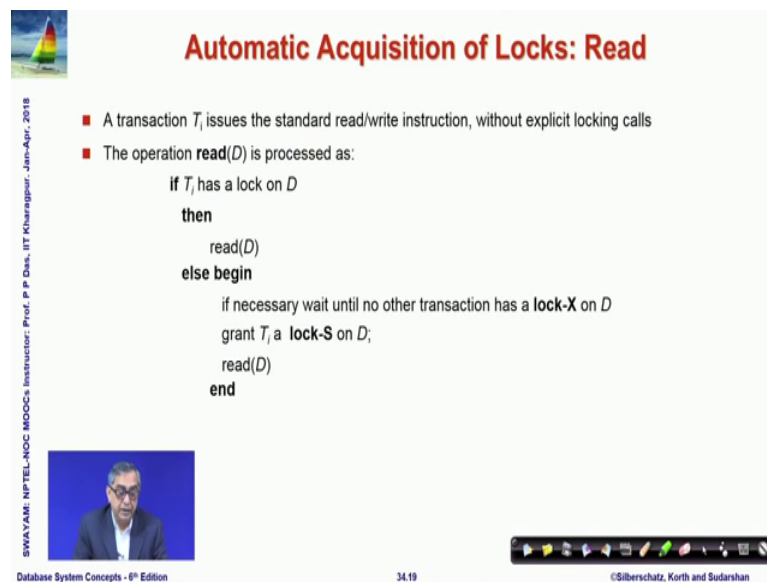
Database System Concepts - 6th Edition 34.18 ©Silberschatz, Korth and Sudarshan

So, you could also in refine the 2 phase locking with what is known as lock conversion that is you can acquire in the in the growing phase, or the first phase you can acquire a

exclusive lock, a shared lock, or you can convert a shared lock that you already have into an exclusive lock which is called the lock upgrade process.

Similarly, in the shrinking phase you can release a shared lock release an exclusive lock, or you are holding an exclusive lock you can make it a shared lock. So, you can download. So, you can understand that upgrade and downgrade are strategies to only use that much of restriction that you need, to impose on others and to allow others to access the data to the based possible way. This protocol again issuers serializability and the it certainly depends on the programmer as to how the programmer inserts the various locking instructions.

(Refer Slide Time: 23:46)



Automatic Acquisition of Locks: Read

- A transaction T_i issues the standard read/write instruction, without explicit locking calls
- The operation $\text{read}(D)$ is processed as:

```
if  $T_i$  has a lock on  $D$ 
then
  read( $D$ )
else begin
  if necessary wait until no other transaction has a lock-X on  $D$ 
  grant  $T_i$  a lock-S on  $D$ ;
  read( $D$ )
end
```

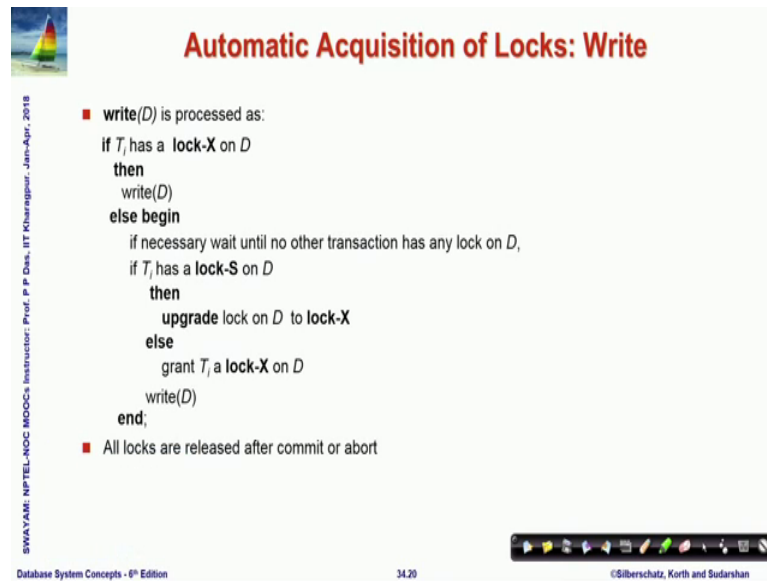
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 34.19 ©Silberschatz, Korth and Sudarshan

Now, you will have to when you want to do read or write, you may acquire locks automatically the database systems will allow that. So, this is a very simple algorithm. So, if you want to read a data, I if you have a lock already on that either shared, or exclusive you can simply read it, if you do not have that then if you may have to wait until no other transaction has an exclusive lock on that because, you know that read or shared lock is not compatible with the exclusive lock.

So, you may have to wait till all are the in no other transaction the transaction that was having exclusive lock possibly has released it and, then take a grant of the shared lock on this item and, then read it is a very simple algorithm to automatically acquire locks.

(Refer Slide Time: 24:37)



Automatic Acquisition of Locks: Write

- `write(D)` is processed as:
 - if T_i has a **lock-X** on D
 - then
`write(D)`
 - else begin
 - if necessary wait until no other transaction has any lock on D ,
 - if T_i has a **lock-S** on D
 - then
upgrade lock on D to **lock-X**
 - else
grant T_i a **lock-X** on D
 - `write(D)`
- end;
- All locks are released after commit or abort

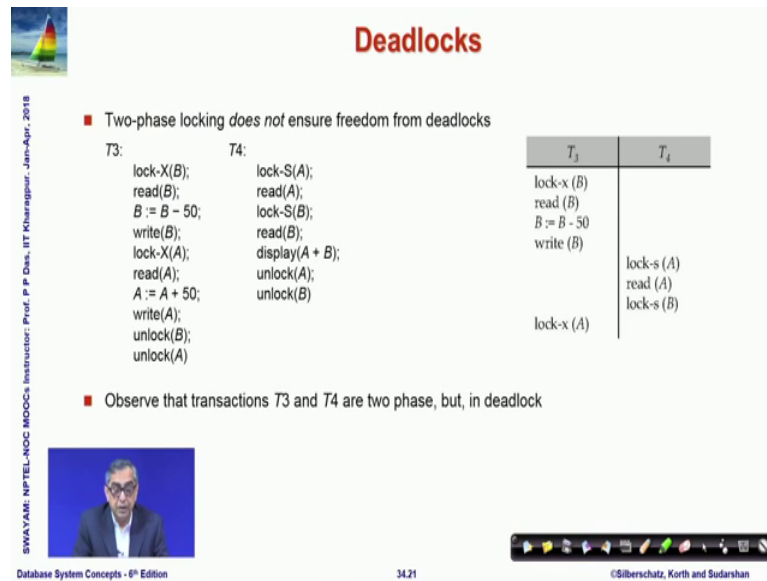
SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 9th Edition 34.20 ©Silberschatz, Korth and Sudarshan

Write is little bit more complex because to be able to write either, you already have an exclusive lock on D , then you write or you may have to wait till no other transaction has any log because, exclusive lock is not compatible with shared lock or with other exclusive lock.

So, as long as some transaction has a lock on D you cannot proceed, but once you come to a state, that you already if no other transaction has a lock, then you see whether you yourself have a shared lock on D , if you have a shared lock then you upgrade it to an exclusive lock, if you do not have a shared lock, then you they take a grant of the exclusive lock and then you can go and write. So, it is if you follow the 2 phases these algorithms become very simple. And when you commit or the abort the transaction, then naturally all locks are get will get released.

(Refer Slide Time: 25:32)



Deadlocks

■ Two-phase locking *does not* ensure freedom from deadlocks

T3: lock-X(B);
read(B);
B := B - 50;
write(B);
lock-X(A);
read(A);
A := A + 50;
write(A);
unlock(B);
unlock(A)

T4: lock-S(A);
read(A);
lock-S(B);
read(B);
display(A + B);
unlock(A);
unlock(B)

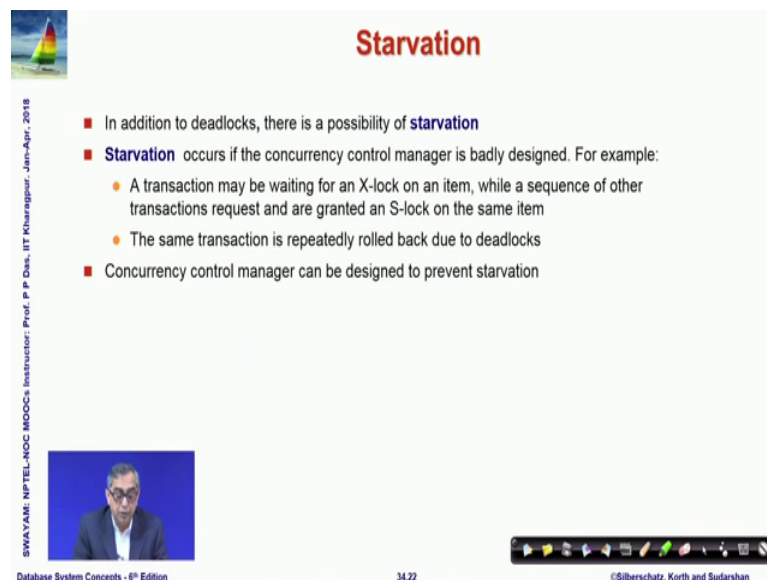
T ₃	T ₄
lock-x (B)	
read (B)	
B := B - 50	
write (B)	
	lock-s (A)
	read (A)
	lock-s (B)
lock-x (A)	

■ Observe that transactions T3 and T4 are two phase, but, in deadlock

Database System Concepts - 6th Edition 34.21 ©Silberschatz, Korth and Sudarshan

So, the 2 phase protocol we have already seen that does not ensure freedom from deadlock, you can may follow 2 phase locking protocol here is an example, but you may still have schedules which will have deadlocks. So, this is one example you can just convince yourself.

(Refer Slide Time: 25:52)



Starvation

■ In addition to deadlocks, there is a possibility of **starvation**

■ **Starvation** occurs if the concurrency control manager is badly designed. For example:

- A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item
- The same transaction is repeatedly rolled back due to deadlocks

■ Concurrency control manager can be designed to prevent starvation

Database System Concepts - 6th Edition 34.22 ©Silberschatz, Korth and Sudarshan

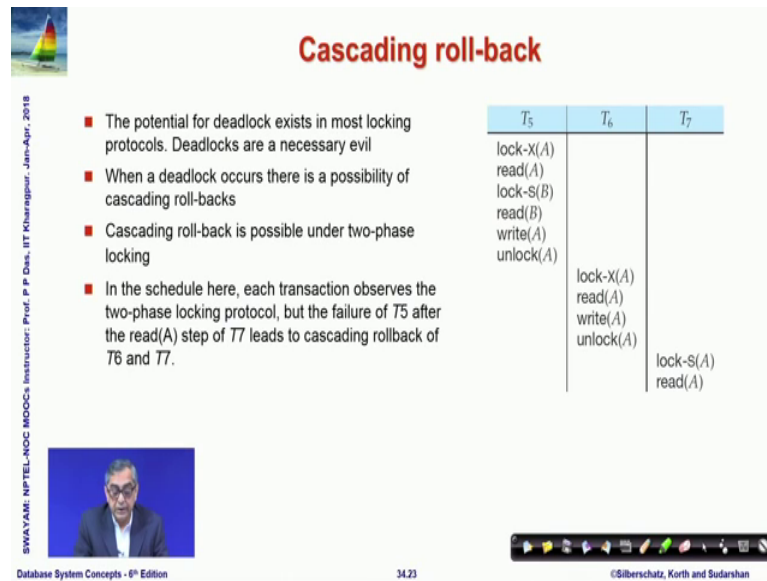
There is another problem that can happen, in addition to deadlock this is a code there is a possibility of what is known as starvation; starvation, occurs usually it occurs when the control concurrency control manager is not a efficient 1.

So, what did we see in terms of automatic locks in read and write operation is you may have to wait because, someone else is holding a lock on an item. Now holding an exclusive lock on the item, now it is possible that like the current transaction there may be couple of other transactions who are also waiting for a lock on that item and, when the opportunity comes that there is no log being held by any transaction, one of the waiting transactions must be given the lock you cannot if it is an exclusive lock you cannot give it to more than 1 transaction, but say 3 transactions were waiting for the exclusive lock and one of them get, that and that transaction can proceed the other transactions have to rollback because, they are not getting the lock.

So, now you again start you again come to the point where you wanted the exclusive lock on that item and at that time somebody is holding it and there are other transactions who are also requesting for exclusive lock. And when you come back and when finally, the exclusive lock is released by all other transactions, then again it is possible that while you are waiting some other transaction that was waiting who gets that exclusive lock and you do not get that so, you roll back and this could repeatedly could keep on happening.

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

(Refer Slide Time: 27:57)



Cascading roll-back

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil
- When a deadlock occurs there is a possibility of cascading roll-backs
- Cascading roll-back is possible under two-phase locking
- In the schedule here, each transaction observes the two-phase locking protocol, but the failure of T_5 after the read(A) step of T_7 leads to cascading rollback of T_6 and T_7 .

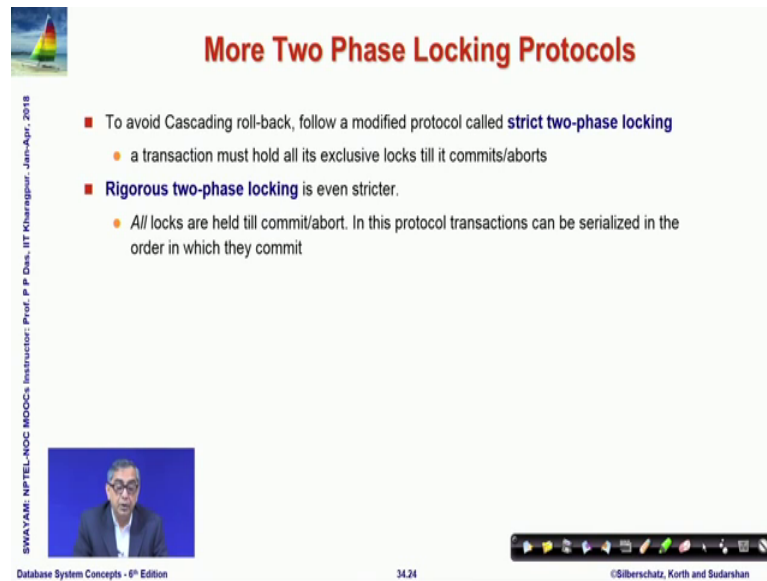
T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

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

Database System Concepts - 6th Edition 34.23 ©Silberschatz, Korth and Sudarshan

There is a the potential for deadlock exists in most locking protocols, as we have seen and when a deadlock occurs there is a possibility of cascading roll back because, when it deadlock happens then naturally you will have to roll back. So, you may have to do a cascading roll back as this example is showing. And it is possible for a 2 phase locking protocol have we have in the example is shown here, where all the transactions are following cascading roll back has to as following 2 phase locking protocol, but if T_5 fails after the read step of T_7 after the read step of T_7 , if T_5 fails then it leads to a cascading rollback T_7 T_5 has to be rolled back. So, T_6 will have to be rolled back, so T_7 will have to be rolled back and so on. ah

(Refer Slide Time: 28:54)



The slide is titled "More Two Phase Locking Protocols" in red text. It features a small image of a sailboat in the top left corner. The main content is a bulleted list of protocols:

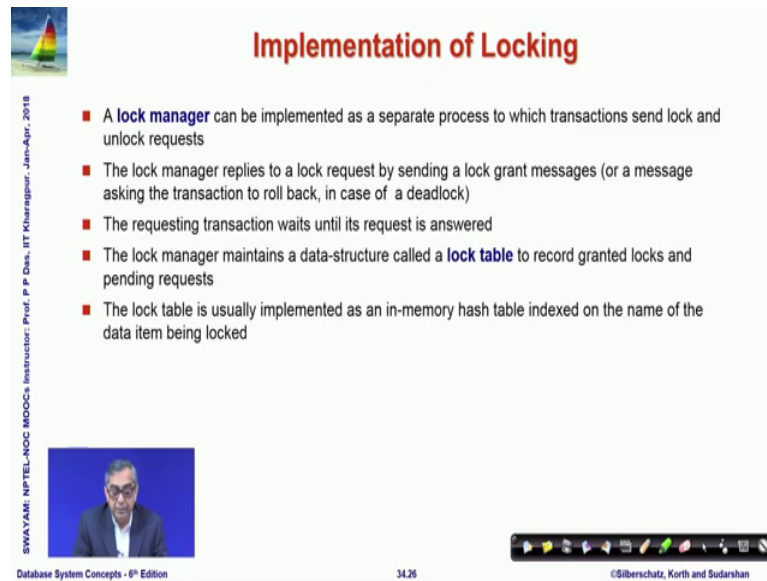
- To avoid Cascading roll-back, follow a modified protocol called **strict two-phase locking**
 - a transaction must hold all its exclusive locks till it commits/aborts
- **Rigorous two-phase locking** is even stricter.
 - All locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit

At the bottom left, there is a small video inset showing a man speaking. The slide footer includes the text "SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018" on the left, "Database System Concepts - 6th Edition" and "34.24" in the center, and "©Silberschatz, Korth and Sudarshan" on the right.

Interestingly there are several other protocols and particularly 2 more 2 phase locking protocol 1 is called strict 2 phase locking, which avoids cascading roll back, where a transaction must hold all exclusive locks till it finally, commits and aborts naturally you can figure out that you are making the time for the transaction to hold lock longer. So, naturally the level of concurrency will go down that is all possible serializable schedules will be smaller, but this guarantees that you will not have a cascading roll back. And there is an even stricter rigorous 2 phase locking where all locks are held till commit or abort.

In the strict 1 only exclusive locks are held till commit or abort there is a till the end of that transaction, but in rigorous 2 phase locking all locks are held till the committed abort, in this protocol transaction can be serialized, in the order in which they do the commit and in that way this is a serializable protocol, which also avoids the cascading roll back up. Now finally, before you close this module a quick word in terms of how do you implement locking.

(Refer Slide Time: 30:09)



Implementation of Locking

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked

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

Database System Concepts - 6th Edition 34.26 ©Silberschatz, Korth and Sudarshan

It is the lock there is a lock manager, which implements the locking the lock manager itself runs on a different process to which every transactions end lock and unlock requests. And the lock manager maintains a data structure to maintain what are the transactions, who are holding different locks on different items and based on that the grant messages are queued on that data structure and, these messages actually release the locks and, otherwise the transaction has to wait the lock manager maintains this as a lock table. And this is typically a in memory hash table because, it needs to naturally be very fast and is in the name of the data item being locked.

(Refer Slide Time: 31:01)

Lock Table

- Dark blue rectangles indicate granted locks; light blue indicate waiting requests
- Lock table also records the type of lock granted or requested
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted
- If transaction aborts, all waiting or granted requests of the transaction are deleted
- lock manager may keep a list of locks held by each transaction, to implement this efficiently

Legend:
■ granted (dark blue)
■ waiting (light blue)

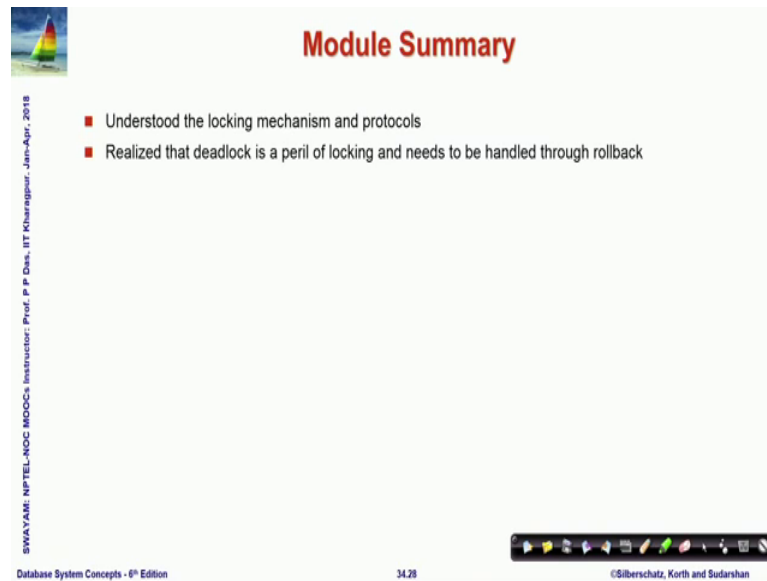
SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 6th Edition 34.27 ©Silberschatz, Korth and Sudarshan

So, let us just show you and so, these are the different this is an instance of a lock table and, the nodes are different data items. So, I 7, I 9, 12, I 4, I 44, I 23 are different data items this is a hash table. So, you can see that on I 7 and I 23 there is a collision and there is collate state chain happening on that. And then for every item you have you maintain a list of locks that are granted to different transactions and the list of requests that are waiting, the dark blue here shows the grant and the light blue shows a waiting status here.

So, it takes it naturally says what type of lock o'clock is granted and requested and based on this therefore, when you get a request to put it in the you come and put it you hash it to that data item, put that request on that queue and based on the current status you can decide, whether it can be granted or it has to wait. So, it is added new requests are added at the end of that queue, it is first in first out and whenever a release happens, then naturally a granted node is removed and a waiting node might get a chance to block that item, if the transaction reports all waiting, or granted requests of the transactions certainly will get deleted ok.

(Refer Slide Time: 32:30)



The slide is titled "Module Summary" in red text. It features a small image of a sailboat in the top left corner. Below the title, there are two bullet points, each preceded by a red square. The first bullet point reads "Understood the locking mechanism and protocols" and the second reads "Realized that deadlock is a peril of locking and needs to be handled through rollback". On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018". At the bottom of the slide, there is a footer with three items: "Database System Concepts - 9th Edition" on the left, "34.28" in the center, and "©Silberschatz, Korth and Sudarshan" on the right. A navigation bar with various icons is located at the bottom right of the slide area.

Module Summary

- Understood the locking mechanism and protocols
- Realized that deadlock is a peril of locking and needs to be handled through rollback

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

Database System Concepts - 9th Edition 34.28 ©Silberschatz, Korth and Sudarshan

So, this is a simple way to manage the locks. So, in this module on concurrency control we have understood the basic locking mechanism and protocols, we have specifically looked at the lock compatibility matrix and the strategies of granting and releasing locks and, we have seen the consequent danger of having deadlock and in some cases starvation, which we have agreed to live with. So, if deadlock happens we will have to roll back one or more transactions and then restart again and, but we cannot take the risk of not having serializable transactions because, that might lead to inconsistent state of the database which is not acceptable.