

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

Lecture – 32
Transactions/2: Serializability

Welcome to module 32 of Database Management Systems, from the last module we are discussing about transactions and transaction management.

(Refer Slide Time: 00:27)

PPD

Module Recap

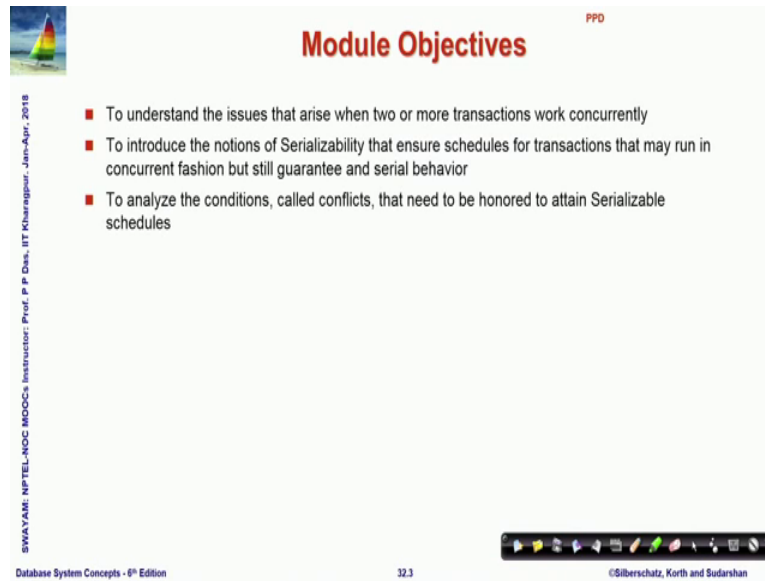
- Transaction Concept
- Transaction State
- Concurrent Executions

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

Database System Concepts - 6th Edition 32.2 ©Silberschatz, Korth and Sudarshan

And we have technical look into the basic concept of a transaction the transaction state and the issues.

(Refer Slide Time: 00:35)



PPD

Module Objectives

- To understand the issues that arise when two or more transactions work concurrently
- To introduce the notions of Serializability that ensure schedules for transactions that may run in concurrent fashion but still guarantee serial behavior
- To analyze the conditions, called conflicts, that need to be honored to attain Serializable schedules

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

Database System Concepts - 6th Edition

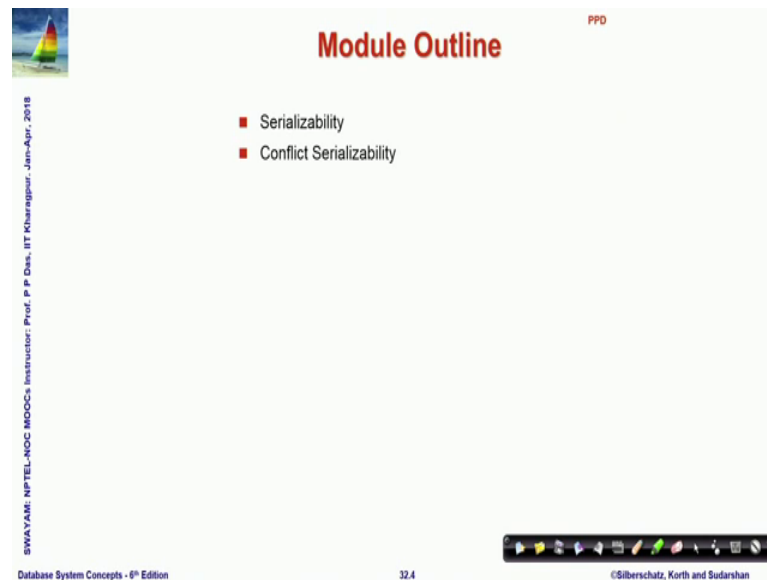
32.3

©Silberschatz, Korth and Sudarshan

In concurrent execution and in this module, we will look try to understand, what are the very specific issues that happen when 2 or more transactions work concurrently we have seen that now it is possible that they execute in a schedule, which would not let us preserve the acid properties.

So, we want to introduce the very basic concept of making sure that such concurrent execution schedules are acceptable, and those are the notions of serializability. And we will analyze different conditions called conflicts that need to be honored to attend the serializability of the schedules.

(Refer Slide Time: 01:17)



PPD

Module Outline

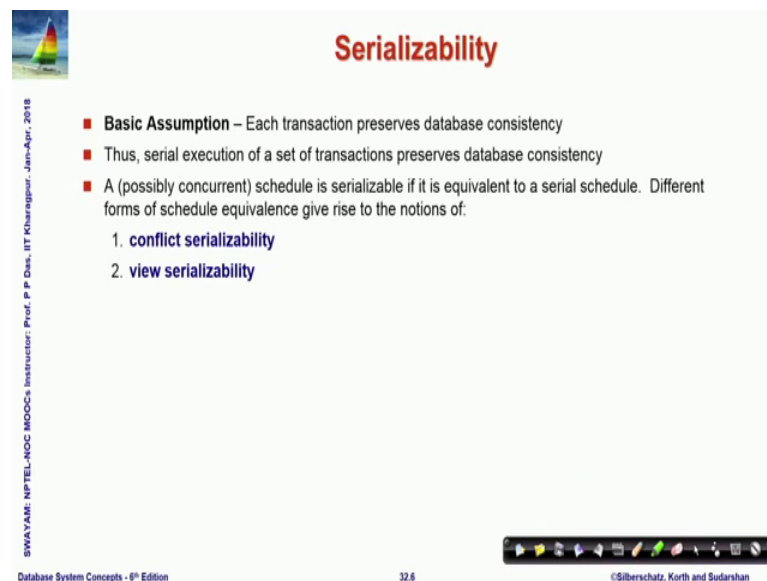
- Serializability
- Conflict Serializability

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

Database System Concepts - 6th Edition 32.4 ©Silberschatz, Korth and Sudarshan

So, serializability is the main topic to discuss.

(Refer Slide Time: 01:21)



Serializability

- **Basic Assumption** – Each transaction preserves database consistency
- Thus, serial execution of a set of transactions preserves database consistency
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
 1. **conflict serializability**
 2. **view serializability**

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

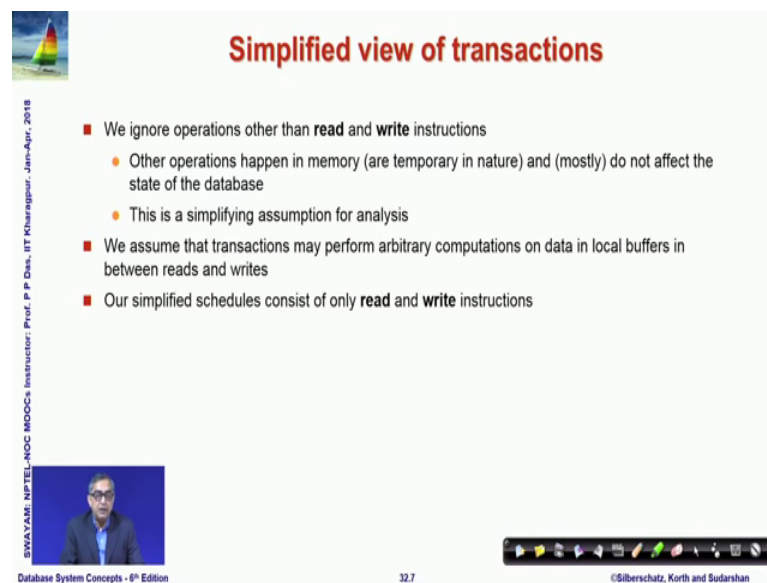
Database System Concepts - 6th Edition 32.6 ©Silberschatz, Korth and Sudarshan

So, to understand serializability we make a basic assumption, we make an assumption that every transaction by itself preserves the database consistency. That is, it starts in a consistent state of the database. And through the execution of its instructions in the order given it leaves the database in a consistent state, that is satisfied in each and every transaction. So, we can conclude that, if we really execute a set of instruction set of transactions, then the consistency of the database will always be preserved.

Now, the problem happens, and as we have seen in the last module, that problems happen when possibly concurrent transactions happen. And we may execute may be executing the instruction in an order which leads to the violation of acid properties, the consistency in particular. So, we say that a concurrent schedule is serializable, if there is a there is some serial schedule, you say what is the serial schedule serial schedule is where the transactions are executed one after the other.

So, if you have 2 transactions in the concurrent system, then if I do T 1 then I do T 2 it is a serial schedule. If I do T 2 and then I do T 1 it is a serial schedule as well. So, if I have a concurrent schedule, like few refer back to the last module schedule 3; where the instructions of T 1 and T 2 are interleaved, then it is it will have to be equivalent to a serial schedule either T 1 after T 2 or T 2 after T 1. Different forms of schedule equivalence is used one is called conflict serializability, and the other is called view serializability. In the present module we will first discuss conflict serializability.

(Refer Slide Time: 03:19)



Simplified view of transactions

- We ignore operations other than **read** and **write** instructions
 - Other operations happen in memory (are temporary in nature) and (mostly) do not affect the state of the database
 - This is a simplifying assumption for analysis
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes
- Our simplified schedules consist of only **read** and **write** instructions

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

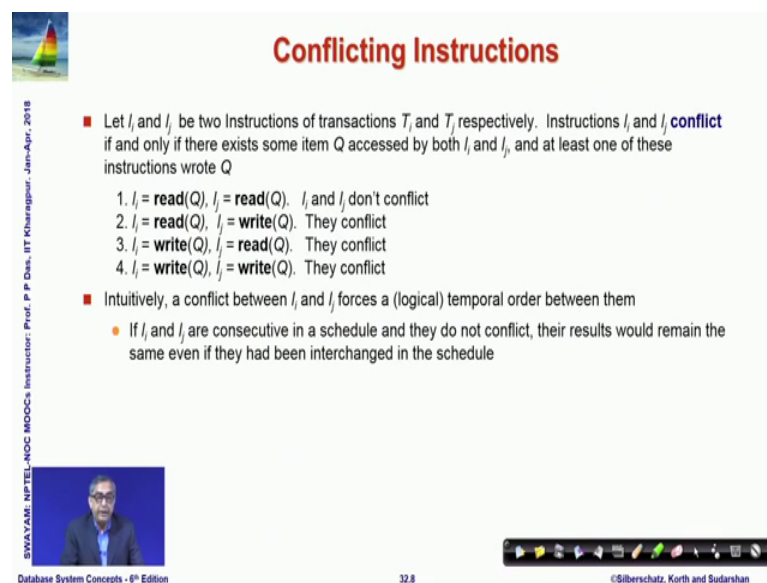
Database System Concepts - 6th Edition 32.7 ©Silberschatz, Korth and Sudarshan

Now, we make now a transaction may have all varied kinds of instructions, but we make an assumption that we will ignore anything other than any instruction other than the read and write instruction. Because other operations like we saw an operation where an account is debited by 50 or account is credited. So, you subtract 50 you add 50 you multiply by point one or things like that are all operations that happen in the local buffer in the memory, and never temporary in nature and mostly they do not affect the state of

the database, because you have read the data do the changes write it back. So, it is a read and write that actually are important for that maintaining the consistency after database. So, that simplifies our process of analysis to a good extent.

So, this is so, we assume that between every read and write or read and read write and write and so on, the database the transactions may be doing arbitrary computations, which are all in the local buffer and do not affect the state. So, we can make this assumption that our shift schedules consists only of read and writing.

(Refer Slide Time: 04:31)



Conflicting Instructions

- Let I_i and I_j be two Instructions of transactions T_i and T_j respectively. Instructions I_i and I_j **conflict** if and only if there exists some item Q accessed by both I_i and I_j , and at least one of these instructions wrote Q
 1. $I_i = \text{read}(Q), I_j = \text{read}(Q)$. I_i and I_j don't conflict
 2. $I_i = \text{read}(Q), I_j = \text{write}(Q)$. They conflict
 3. $I_i = \text{write}(Q), I_j = \text{read}(Q)$. They conflict
 4. $I_i = \text{write}(Q), I_j = \text{write}(Q)$. They conflict
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them
 - If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule

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

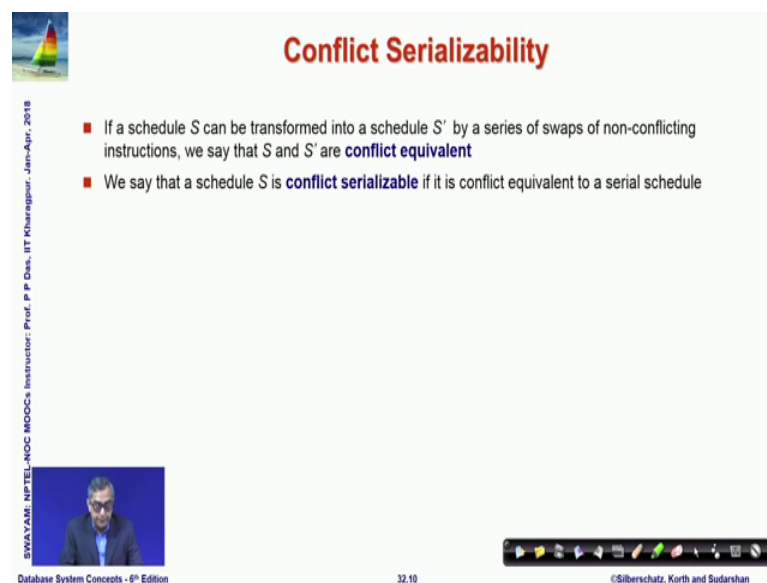
Database System Concepts - 6th Edition 32.8 ©Silberschatz, Korth and Sudarshan

Now, we say that suppose I_i and I_j , 2 instructions for belonging to transaction T_i and transaction T_j . So, there are 2 transactions T_i and T_j , T_i has an instruction I_i T_j has an instruction I_j and we say that I_i and I_j this instruction will conflict, if and only if there is some item Q , that is some data entity Q ; which both I_i and I_j are trying to access. And at least one of these instructions try to write.

So, these 2 instructions from true transactions are trying to manipulate the same data item, and at least one of them is trying to write. If that happens then we say that I_i and I_j these 2 instructions are conflicting. So, you can naturally enumerate the 4 possibilities, if both of them are reading their own conflict. If it is read write, write read, write All of them are cases of conflict.

So, naturally intuitively, you can figure out that since the write changes are value that if there is a conflict between these 2 instructions then there must be a fixed temporal order between them. So, if I_i and I_j are consecutive in a schedule and they do not conflict. Then we can interchange the temporal order of I_i and I_j , that will also not make a difference, because they do not conflict. But if they conflict I cannot make the change in their ordering.

(Refer Slide Time: 06:18)



Conflict Serializability

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**
- We say that a schedule S is **conflict serializable** if it is conflict equivalent to a serial schedule

Database System Concepts - 6th Edition 32.10 ©Silberschatz, Korth and Sudarshan

So, that gives rise to the notion of conflict serializability. So, we say if a schedule S can be transformed into another schedule S' by a series of swaps of non-conflicting instructions, then S and S' are conflict equivalent. So, what are you saying? That we have 2 one schedule S , and we will swap non-conflicting instruction, possibly since non-conflicting instructions that occur side by side. And if by doing this, if I can create the schedule S' , then I will say S and S' are conflict equivalent. But if S and S' are such that, I cannot transform S into S' by just swapping non-conflicting instructions, then they are not conflict equivalent.

The second definition to keep in mind is a schedule S is conflict serializable, if it is conflict equivalent to a serial schedule, what is the serial schedule? Just to remind you serial schedule is one where the transactions are happened one after the other in a serial manner. So, all instructions of one transaction complete, then all instructions of the second transaction complete, then all instructions of the third transaction complete and so

on. So, if a schedule is conflict serializable; that is, if in a schedule. I can swap non-conflicting instructions. And make it into a serial schedule, and then I will say that the given schedule is a conflict serializable schedule ok.

(Refer Slide Time: 08:00)

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6 – a serial schedule where T_2 follows T_1 , by a series of swaps of non-conflicting instructions.
 - Swap $T_1.read(B)$ and $T_2.write(A)$
 - Swap $T_1.read(B)$ and $T_2.read(A)$
 - Swap $T_1.write(B)$ and $T_2.write(A)$
 - Swap $T_1.write(B)$ and $T_2.read(A)$

These swaps do not conflict as they work with different items (A or B) in different transactions.

Therefore, Schedule 3 is conflict serializable:

T_1	T_2	T_1	T_2	T_1	T_2
read(A)		read(A)		read(A)	
write(A)		write(A)		write(A)	
	read(A)		read(A)		read(A)
	write(A)		write(A)		write(A)
		read(B)		read(B)	
		write(B)		write(B)	
read(B)			read(B)		
write(B)			write(B)		
	read(B)		read(B)		read(B)
	write(B)		write(B)		write(B)

Schedule 3 is circled in red. Schedule 6 is also circled in red.

So now let us it is time for a number of examples to understand this better. So, we had seen schedule 3, will have to refer to the earlier module 4 schedule 3. Sir, no not I am sorry this is just abstracted form of that; not the actual one because in the in the earlier schedule 3 we had shown all the complete other computations also, but the read writes are the same.

Now, that this schedule 3 can be converted to so, this is where you have schedule 3, and you can easily see that the part of transaction T_1 then a part of transaction T_2 . So, schedule 3 is not a serial schedule, but if you can swap non conflicting instructions, then you are able to convert this into this schedule which if we are calling a schedule 6. Where all instructions of T_1 is followed by all instructions of T_2 which is a serial schedule.

So, since this can be done, we will say it is conflict serializable schedule 3 is conflict serializable and just to see how that happens. So, you start here let me erase this marks and start here. So, here if I look into these 2 instructions, which are the consecutive instructions in schedule 3 I can swap them; that is, I can do read B first and then do read A, I can swap read B and write A read B, and write A can be swapped.

Once I have done that, then I can swap read B with read A. It has become before right I can swap it with, because read B and write A, or write B read B and read A these do not conflict their non-conflicting instruction. Why read B an righty and non-conflicting, because they are not reading and writing to the same data item. Why read B and read A are non-conflicting, they are accessing the same data item, but both of them are read there is no right. So, I can swap so, this is the second one I can. So, once I do that read B will come here and write A read A write A will come down.

Then again, I can see that write B can be swapped with write A. Both are rights, but referring to different data items. Similarly, write B then can be swapped with read A, because they are again referring to different data items. So, I can do this and then these will also come up. So, I will eventually after these 4 swaps, this whole schedule 3 will transform into this serial 6, and we get a serial schedule.

So, we will say that schedule 3 is conflict serializable. That is the basic concept that we are trying to establish here.

(Refer Slide Time: 11:02)

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

T_3	T_4
read (Q)	write (Q)
write (Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$, or the serial schedule $\langle T_4, T_3 \rangle$

Database System Concepts - 6th Edition | 32.12 | ©Silberschatz, Korth and Sudarshan

Just as very simple example suppose you had 2 transactions T 3 and T 4, and you have this situation. Now is it conflict serializable it is not. Because to make it conflict serializable. I need to either swap right Q of T 3 with right Q of T 4 which is not possible because these are conflicting instructions, they both access the same data item Q and they both are right.

The other option is I could swap read Q in T 3 and write Q in T 4, that they are also conflicting because they access the same data item and one of them is right. So, I cannot do either of this swaps which mean, that I cannot find a conflict equivalent schedule for this schedule; either to T 3 T 4 or to T 4 T 3. It is not this schedule is not conflict equivalent to either one of them.

So, this schedule is not conflict serializable, this is the core concept. So, if you if you go through different examples and try to understand this at the very beginning, then in terms of the transaction management the whole study of transaction management you will have very easy progress.

(Refer Slide Time: 12:37)

Example: Bad Schedule

Consider two transactions:

Transaction 1	Transaction 2
UPDATE accounts	UPDATE accounts
SET balance = balance - 100	SET balance = balance * 1.005
WHERE acct_id = 31414	

In terms of read / write we can write these as:

Transaction 1: $r_1(A), w_1(A)$ // A is the balance for acct_id = 31414
 Transaction 2: $r_2(A), w_2(A), r_2(B), w_2(B)$ // B is balance of other accounts

Consider schedule S:

- Schedule S: $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
- Suppose: A starts with \$200, and account B starts with \$100
- Schedule S is very bad! (At least, it's bad if you're the bank!) We \$100 from account A, but somehow the database has that our account now holds \$201!

	A	B
(initial:)	200.00	100.00
$r_1(A)$:		
$r_2(A)$:		
$w_1(A)$:	100.00	
$w_2(A)$:	201.00	
$r_2(B)$:		
$w_2(B)$:		100.50

Schedule S

Source: <http://www.cburch.com/cs/340/reading/serial/>

Database System Concepts - 6th Edition 32.13 ©Silberschatz, Korth and Sudarshan

So, let us let me show you number of other bad schedules, and let me a little bit more complex examples.

So, consider 2 transactions transaction 1 here. Update an account, where the account id is 31414 a specific account and balance is debited by 100. So, it is debiting 100 from the balance. Where in the transaction 2, you update accounts where balance is changed to balance times 1.005 which means that we are giving a point 5 percent interest, and here there is no where clause. So, transaction 2 actually changes does this balance change in all the accounts, whereas, transaction 1 makes this debit in only one account.

Let see what will happen in terms of them. So, let us first try to write out transaction 1 and transaction 2, the first in the read write Abstracted form. So, transaction 1 it is working only on one account let us call it account A. So, what does it do? It has to set the balance to debit 100. So, it has to read so, this is r_1 by $r_1 A$, we mean that it is read the subscript here refers to the transaction number.

So, r_1 stands for r stands for read, 1 stands for transaction 1. So, it is read by transaction 1. And what are we reading? We are reading the account balance A, let us arbitrarily we are calling it A. And then what we will have to do? After having debited that locally we will have to write it back so that the change has happened. So, $r_1 A$ followed by $w_1 A$ is transaction 1 which is being shown on the left. So, I have shown you from the actual sql statement, how can you make an abstraction of the read write that we use in terms of reasoning about the serializability.

In contrast, if you look at transaction 2, naturally transaction 2 does not have a where clause. So, it performs this balance update on each of the accounts. So, it will also perform this balance update on the account A or account balance a rather, that we assumed in the transaction 1. So, we model that by saying that naturally for changing the balance from balance times 1.005, we need to read A if the read is done in transaction 2. So, that is $r_2 A$ and write it back. So, that is $w_2 A$ and then I assume that B is some other account. There may be one more account there may be 100 thousand more account, but so far as serializability are concerned, these are all other different accounts from A. So, we symbolically just consider one; that is, some other account B other than the balance a and naturally to do the change here or do the update here will have to read B r to be and w to B. So, these are the 2 transactions in the simplified form that we have to analyze.

Now, let us consider A so, we have between transaction 1 and transaction 2 we have 6 instructions. So, we produce a schedule 6, where these 6 instructions are interleaved. And we satisfy the basic constraint that the instructions of every transaction occur in the same order in which they existed. So, r_1 precedes w_1 r_1 precedes w_1 in this schedule. $R_2 A$ precedes $w_2 A$, $w_2 A$ precedes $r_2 B$, $r_2 B$ precedes $w_2 B$ and so on. So, their original ordering is maintained, but we have an interleaved schedule called S. And then on the on the write if you if you look at here on the write this is schedule S.


So, in the write we are saying, that let us say that a starts with 200 dollar, and B at the beginning is 100 dollar. So, what will happen you will read? You will read here this is the first thing r 1 A. So, 200 is read then r 2 A. So, what happens if r 2 A is read Again 200 is read. And then you do w 1 A. So, what is w 1 A? W 1 A is the right of the transaction 1. So, transaction 1 writes after debiting this balance this is the intermediate computation.

So, when transaction 1 writes, it writes based on the value that it had read in r 1 A; which was 200 then 100 debited. So, it writes back 100. Then the next is w 2 A. So, w what will w 2 A do w 2 A will write back the write back w 2 A is here, will write back the result of the computation in transaction 2 based on what it read in the r 2 A. R 2 A had written had read 200, we hear and therefore, if you multiply 200 by this factor it becomes 201. So, w 2 A will write 201.

So, naturally E as w 2 A has changed the value of a after w 1 A naturally the final value of a will be 201. Then you have r to be w to be which reads 100 makes this balance change by 1.005, it becomes 100.5. So, this is what we will have when actually this schedule completes. So, if I mean just this look at what has happened, it has I have debited 100 dollar from account A which was transaction 1, bar and here I had started with 200 dollar. But at the end what I have according to the schedule is account A has a balance which is 201 dollar. Whereas, it should have had a balance which should have been 100 dollar, the balance in account B is fine. But it shows that 101 dollar more in account A.

So, naturally the bank is going to get bankrupt very soon if such scheduled are allowed. So, this schedule is an incorrect inconsistent schedule, it is a bad schedule, let us take other examples.

(Refer Slide Time: 19:18)



Example: Bad Schedule

- Ideal schedule is serial: (A = \$200, B = \$100)
- Serial schedule 1: $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B) // A = 100.50, B = 100.50$
- Serial schedule 2: $r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A) // A = 101.00, B = 100.50$
- We call a schedule **serializable** if it has the same effect as some serial schedule regardless of the specific information in the database.
- As an example, consider Schedule T, which has swapped the third and fourth operations from S:
 - Schedule S: $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
 - Schedule T: $r_1(A), r_2(A), w_2(A), w_1(A), r_2(B), w_2(B)$
- By first example, the outcome is the same as Serial schedule 1. But that's just a peculiarity of the data, as revealed by the second example, where the final value of A can't be the consequence of either of the possible serial schedules.

	A is \$100 initially		A is \$200 initially	
	A	B	A	B
(initial:)	100.00	100.00	200.00	100.00
$r_1(A):$				
$r_2(A):$				
$w_2(A):$	100.50		201.00	
$w_1(A):$	0.00		100.00	
$r_2(B):$				
$w_2(B):$	100.50		100.50	

Schedule T

Source: <http://www.cburch.com/cs/340/reading/serial/>

Database System Concepts - 6th Edition
32.14
©Silberschatz, Korth and Sudarshan

Now let us ask what is the ideal schedule, what is ideally what should happen. Ideally naturally we can have we will have serial schedules, there are 2 transactions. So, there are 2 possible serial schedules that can happen that is first T 1 happens, then hole of T 2 happens. I am sorry, first T 1 and then hole of T 2. Or first hole of T 2 and then T 1. And if you go through the steps, assuming that a initially is 200 dollar and B is 100 dollar, these are the possible results that you see naturally. As I had mentioned earlier also, the different ordering different schedule might give you different results, but both of them are correct, because any one of them will happen, but both are consistent. Either debit has first happened, then the interest rate or interest rate it has first happened and then the debit.

So, either of these schedules are acceptable, but what we got as a schedule S in the last case are not acceptable. So, we will call it you will serializable, if it has the same effect, as some of the one of the 2 schedules that we have here. Then we will say that is it this is serializable schedule. So, again we create another example schedule T here. So, what we do? We take the schedule S which we saw was bad, and we interchange, these 2 we do w 2 for a first and w 1 A next.

Now, you see very interesting things will happen. So now, you focus on this part, on the left part of schedule T where we are assuming that A and B both have 100 dollar to start with. And then go through these steps r 1 is in transaction 1 r 2 is in transaction 2, then w

2 happens so, the interest is credit 100.5. And then what has happened? W 2 after that w 1 A so, whatever was written here is debited and written back. So, whatever was read there is 100 dollar. So, you debit 100 dollar it becomes 0.

So, a has become 0, and then you have the B which goes on correctly. So, things look like that it appears that we are we are perfectly ok. So, by the first example the outcome is same as a serial schedule one. And so, we might just think that things have been good, but this is just incidental based on the particular values. Now let us consider another execution by the same schedule which makes use of this value 200 and 100.

Now, as it with 200 and 100 and we do w 2 followed by w 1. So, when r 2 A is followed by w 2 r 2 A 100 read 200 and that 10 1.005 or that kind of interest is given then it becomes 201. And then r 1 then you have w 1. Now what does w 1 A changes? R 1 had read 200, and from that you have subtracted 100. So now, you have as w 1 A, you have 100 input. And from this you have B certainly does not change.

So, if you look into that, now you can see that he has a value which is 100; which certainly if you if you look back. So, 200 and 100 are the values that we had assumed here, and you can see that in neither of the schedule a can have a value, which is 100 dollar as we have found here. It can either be 100.50 or it can be 101, but you have got a value 100. So, even though with some data, a schedule might look like serializable, but it actually is not and it needs to be properly established that serial is serializable.

(Refer Slide Time: 24:00)

PPD

Example: Good Schedule

- What's a non-serial example of a serializable schedule?
 - We could credit interest to A first, then withdraw the money, then credit interest to B:
 - Schedule U: $r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B)$ // A = 101, B = 100.50
- Schedule U is conflict serializable to Schedule 2:

Schedule U:	$r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B)$
swap $w_1(A)$ and $r_2(B)$:	$r_2(A), w_2(A), r_1(A), r_2(B), w_1(A), w_2(B)$
swap $w_1(A)$ and $w_2(B)$:	$r_2(A), w_2(A), r_1(A), r_2(B), w_2(B), w_1(A)$
swap $r_1(A)$ and $r_2(B)$:	$r_2(A), w_2(A), r_2(B), r_1(A), w_2(B), w_1(A)$
swap $r_1(A)$ and $w_2(B)$:	$r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A)$: Schedule 2

Source: <http://www.cburch.com/cs/340/reading/serial/>

So, neither S nor T are serializable, yet another schedule U this again. So, you can see that transaction 1 is happening instruction of transaction 1 is happening somewhere in the middle. With transaction 2 and this is what you get. So, if you if you look back as to earlier case. You will find that this is same as scheduled 201. So, this is same as scheduled 2.

So, again my the data it looks like that this is correct, but we have to actually establish that this is correct. So, we can establish say that by proving that schedule 2 is, I am sorry, schedule 2 is conflict serializable, schedule U is conflict serializable to schedule 2. How we do that? We keep on swapping the non-conflicting instructions. This is one we start with, and we swap w_1 with r_2 B this is possible they are referring to 2 different data items. Then we swap w_1 with w_2 again different data items. Then we swap r_1 with r_2 again different data items and also, they are both of them are read. And finally, we swap r_1 with w_2 r_1 A with w_2 B and we get this, and now you can see that this is transaction 2 followed by transaction 1 which is scheduled 2. Which is indeed a serial schedule and we have been able to transform schedule U into a conflict equivalent schedule 2 which is serial.

So, we will say that while our earlier attempts on schedule S and schedule T were not serializable schedule U is serializable.

(Refer Slide Time: 26:09)

Serializability

- Are all serializable schedules conflict-serializable? No.
- Consider the following schedule for a set of three transactions.
 - $w_1(A), w_2(A), w_2(B), w_1(B), w_3(B)$
- We can perform no swaps to this:
 - The first two operations are both on A and at least one is a write;
 - The second and third operations are by the same transaction;
 - The third and fourth are both on B at least one is a write; and
 - So are the fourth and fifth.
 - So this schedule is not conflict-equivalent to anything – and certainly not any serial schedules.
- However, since nobody ever reads the values written by the $w_1(A)$, $w_2(B)$, and $w_1(B)$ operations, the schedule has the same outcome as the serial schedule:
 - $w_1(A), w_1(B), w_2(A), w_2(B), w_3(B)$

Source: <http://www.cburch.com/cs/340/reading/serial/>

Database System Concepts - 6th Edition 32.16 ©Silberschatz, Korth and Sudarshan

So, naturally all serializable schedules are they conflict serializable. No, for example, here I have given. So, here what we are trying to highlight is a schedule may be serializable, but it may not be conflict serializable. So, conflict serializability is a stronger notion. So, here I have given a small example which I leave to you to go through in detail and understand where it is not possible to show that it is conflict serializable in the sense you cannot there are 3 transactions here w_1 , w_2 and w_3 . And you cannot swap non-conflicting instructions in this schedule and convert it into a serial schedule.

So, serial schedule here will mean, $T_1, T_2, T_3, T_1, T_3, T_2, T_2, T_1, T_3$ like that. Any of the 6 possibilities, you cannot convert this in a conflict equivalent manner to any of those 6 serial schedules. But this actually is a serial schedule, because very interestingly even though there are multiple rights, but in between there are no reads. So, you can you can easily reason, that the values have actually not changed ok. So, this is on the basic notion of serializability.

(Refer Slide Time: 27:36)

Precedence Graph

- Consider some schedule of a set of transactions T_1, T_2, \dots, T_n
- **Precedence graph**
 - A directed graph where the vertices are the transactions (names)
- We draw an arc from T_i to T_j if the two transactions conflict, and T_i accessed the data item on which the conflict arose earlier
- We may label the arc by the item that was accessed
- **Example**

The example graph shows two nodes, T_1 and T_2 , each in a light blue circle. There are two directed edges between them: one from T_1 to T_2 and one from T_2 to T_1 , forming a cycle.

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

Database System Concepts - 6th Edition 32.17 ©Silberschatz, Korth and Sudarshan

Now, the question naturally is how do I detect, if a schedule is serializable. So, the process is to construct a what is called a precedence graph. So, if I have a set of transactions, then I construct a graph is a directed graph where the vertices are the transactions their names. And we will draw an art from T_i to T_j , that is my graph means there will be an edge is a directed edge. If these 2 transactions T_i and T_j are conflicting.

So, if T_i T_j conflict there will edge between that. And the edge will be from T_i to T_j , if T_i access the data item which conflict with T_j . So, if T_i is a head is earlier, then we will draw the arc from T_i to T_j , otherwise it will be from T_j to T_i . And we may also annotate label the arc by the item on which item that is being accessed.

(Refer Slide Time: 28:47)

Testing for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph
 - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
 - That is, a linear order consistent with the partial order of the graph.
 - For example, a serializability order for the schedule (a) would be one of either (b) or (c)

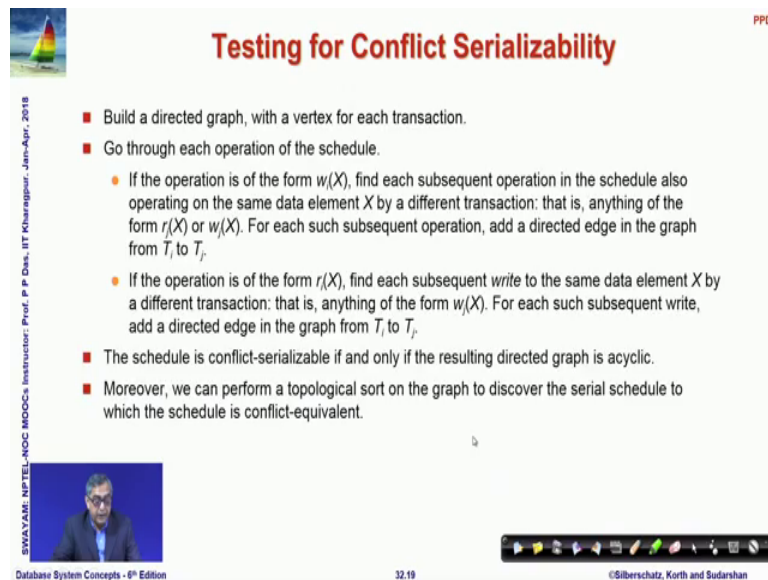
The slide contains three diagrams: (a) a precedence graph with nodes T_i , T_j , and T_k and edges $T_i \rightarrow T_j$, $T_i \rightarrow T_k$, and $T_j \rightarrow T_k$; (b) a linear order T_i, T_j, T_k ; and (c) a linear order T_i, T_k, T_j .

Database System Concepts - 6th Edition
32.18
©Silberschatz, Korth and Sudarshan

So, this could be A so, possible what is called the precedence graph. So, it is a schedule is conflict serializable, if and only if it is precedence graph is acyclic. Naturally, if there is a cycle then; that means, that any of the like we have here, if there if it is a cyclic like this then it is possible that I can actually do a topological ordering of these nodes, and we can find a serial schedule. But if it has a cycle then naturally, I i cannot put any of the transactions on the cycle at the beginning and put the others on the later part things will; obviously, always conflict.

So, we can easily these are details of the algorithms, cycle detection can be done very easily. Either in n square time in a simple manner or when n plus E time where E is a number of edges. So, the precedence if the precedence graph is acyclic the serializability order will be obtained by simple topological sorting. I am not discussing what these algorithms are I would expect that you know if you do not please look up in algorithms book.

(Refer Slide Time: 30:07)



Testing for Conflict Serializability

- Build a directed graph, with a vertex for each transaction.
- Go through each operation of the schedule.
 - If the operation is of the form $w_i(X)$, find each subsequent operation in the schedule also operating on the same data element X by a different transaction: that is, anything of the form $r_j(X)$ or $w_j(X)$. For each such subsequent operation, add a directed edge in the graph from T_i to T_j .
 - If the operation is of the form $r_i(X)$, find each subsequent write to the same data element X by a different transaction: that is, anything of the form $w_j(X)$. For each such subsequent write, add a directed edge in the graph from T_i to T_j .
- The schedule is conflict-serializable if and only if the resulting directed graph is acyclic.
- Moreover, we can perform a topological sort on the graph to discover the serial schedule to which the schedule is conflict-equivalent.

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


Database System Concepts - 6th Edition 32.19 ©Silberschatz, Korth and Sudarshan

So, to test for conflict serializability; the steps will be built the directed graph. Then go through each operation of shall, you look at each operation read or write. If the operation is a write, then find so, if it is $w_i X$, then find what is happening with data this data element X in different transactions that exists later on that instructions exist later on.

If there is some $r_j X$ or some $w_j X$, either in this was in transaction I_i , in transaction some transaction j if there is a read X or if there is a right of X , then there will be a directed graph age from T_i to T_j . This is what I said earlier. On the other case if your operation is of the from $r_i X$ if it is a read operation then all that you need to look for is only a right on this X on the different transaction. And then you will have a naturally if you if your current operation is read and you do not find a write there may be other reads on X then you do not add any conflict edge.

So, on this graph the schedule is conflict serializable if it is acyclic, and we will do topological sort to get that as I have.

(Refer Slide Time: 31:28)

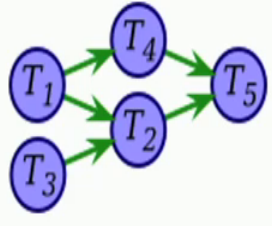


Testing for Conflict Serializability

PPD

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

- Consider the following schedule:
 - $w_1(A), r_2(A), w_1(B), w_3(C), r_2(C), r_4(B), w_2(D), w_4(E), r_3(D), w_5(E)$
- We start with an empty graph with five vertices labeled T_1, T_2, T_3, T_4, T_5 .
- We go through each operation in the schedule:
 - $w_1(A)$: A is subsequently read by T_2 , so add edge $T_1 \rightarrow T_2$
 - $r_2(A)$: no subsequent writes to A, so no new edges
 - $w_1(B)$: B is subsequently read by T_4 , so add edge $T_1 \rightarrow T_4$
 - $w_3(C)$: C is subsequently read by T_2 , so add edge $T_3 \rightarrow T_2$
 - $r_2(C)$: no subsequent writes to C, so no new edges
 - $r_4(B)$: no subsequent writes to B, so no new edges
 - $w_2(D)$: D is subsequently read by T_2 , so add edge $T_2 \rightarrow T_2$
 - $w_4(E)$: E is subsequently written by T_5 , so add edge $T_4 \rightarrow T_5$
 - $r_3(D)$: no subsequent writes to D, so no new edges
 - $w_5(E)$: no subsequent operations on E, so no new edges
- We end up with precedence graph
- This graph has no cycles, so the original schedule must be serializable. Moreover, since one way to topologically sort the graph is $T_3-T_1-T_4-T_2-T_5$, one serial schedule that is conflict-equivalent is
 - $w_3(C), w_1(A), w_1(B), r_4(B), w_4(E), r_2(C), r_2(D), w_5(E)$



Database System Concepts - 6th Edition
32.20
©Silberschatz, Korth and Sudarshan

So, here what I have done is I have actually taken a little bigger example, where you can see that at the beginning here. I have given a schedule which has 5 transactions. And it has A B C D E, 5 different data elements, and variety of read write happening on them. So, based on that, you start with an empty graph having so, your graph will have 5 nodes because these are the transactions. And then you go through the schedule, you start with the very first one $w_1(A)$. So, A is the data item you are looking at and then you see who is doing it. So, you see that well A is read by A is here read by T_2 .

So, there is a conflict. So, you will add an edge $T_1 T_2$ so, this edge gets added. Then is to have $r_2(A)$, and you find look for A, A, A, A, A, A there is no A so, there is no subsequent right. So, there is no new edge then you have $w_1(B)$, $w_1(B)$ and if you look for you have $r_4(B)$; so r_4 that this transaction 4 is reading it later on. So, $w_1(A)$ B subsequently read 5 T_4 so, there is a conflict. So, you add the edge T_1, T_4, T_1, T_4 . You proceed in this way, you can work it out in full. And when you come to the end you have constructed this particular graph which is the precedence graph. And you can very easily see that this precedence graph is acyclic there is no cycle here. And therefore, the original schedule is serializable. And what is that what is the order in which you find out what is the corresponding serial schedule for that you do a topological sort.

So, by topological sort which will mean this have no predecessor. So, any one of them can be the first node other one can be the next node. So, it could be $T_3 T_1$ or $T_1 T_3$.

So, let us say T 3 T 1 then T 3 T 1 has happened. So, I can put any one of T 2 or T 4 after that. Here I put T 4 then T 2. So, up to this and then finally, T 5. So, this is one possible serial schedule to which this given schedule is conflict serializable. And so, the actual serial schedule. So, if you if you do this schedule, you will get a result which is a result of this serial schedule which is T 3, T 1, T 4, T 2, T 5. It is also actually this channel is conflict serializable to several other trade rules because you can do this topological sorting in various different manners, you could have started with T 1 and then do T 3 and then do the rest. You could have done T 3, T 1, and then instead of doing T 4, T 2, you could do T 2, T 4.

So, you will get a number of, but having one equivalent serial schedule. One conflict equivalent serial schedule is enough to prove the serializability of a schedule. Now so, based on that you say that this particular schedule is conflict serializable, and it will be safe to execute the interleaved instructions of the 5 different transactions in this manner in the schedule, and we will always have a consistent result.

(Refer Slide Time: 35:18)

Module Summary

- Understood the issues that arise when two or more transactions work concurrently
- Learnt the forms of serializability in terms of conflict and view serializability
- Acyclic precedence graph can ensure conflict serializability

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

Database System Concepts - 6th Edition 32:21 ©Silberschatz, Korth and Sudarshan

So, here in this module, you have understood the issues that arise in terms of concurrency when 2 or more transactions work concurrently. And very specifically we have learnt about different forms of serializability. In this module we have talked of conflict serializability, view serializability we will take up later on. And we have seen an

algorithm, simple algorithm, based on the a cyclic precedence graph, which will allow you to ensure that a given schedule is conflict serializable or not.