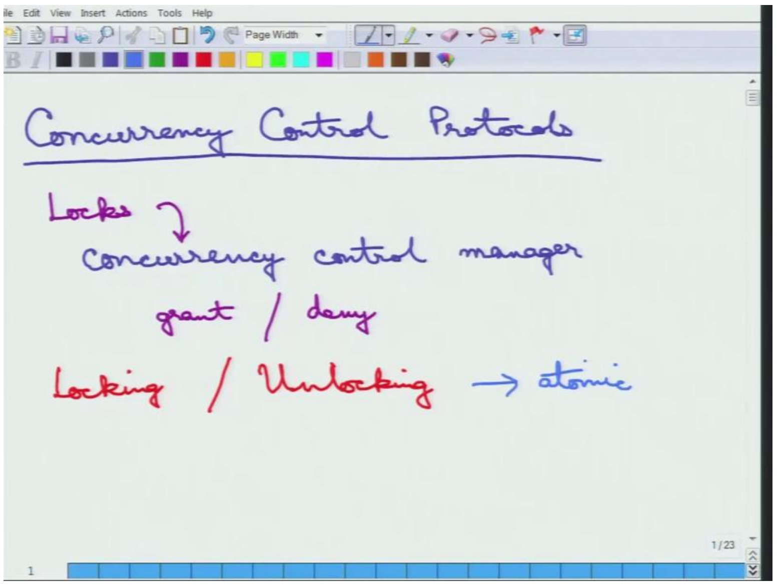
Fundamentals of Database Systems

Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 38
Concurrency Control: Locks

We will continue with the transactions, we have already seen what a transaction is, what a schedule is and some recovery control systems, etcetera. So, we will next move on to concurrency control protocols. So, concurrency control protocols are essentially that is what they decide, how to manage the concurrency between two or more transactions in a schedule. So, the logs, etcetera and when to give access of a particular data item to a transaction that is what we will be studying next.

(Refer Slide Time: 00:42)



Concurrency Control Protocols

Locks → Concurrency control manager

grant / deny

Locking / Unlocking → atomic

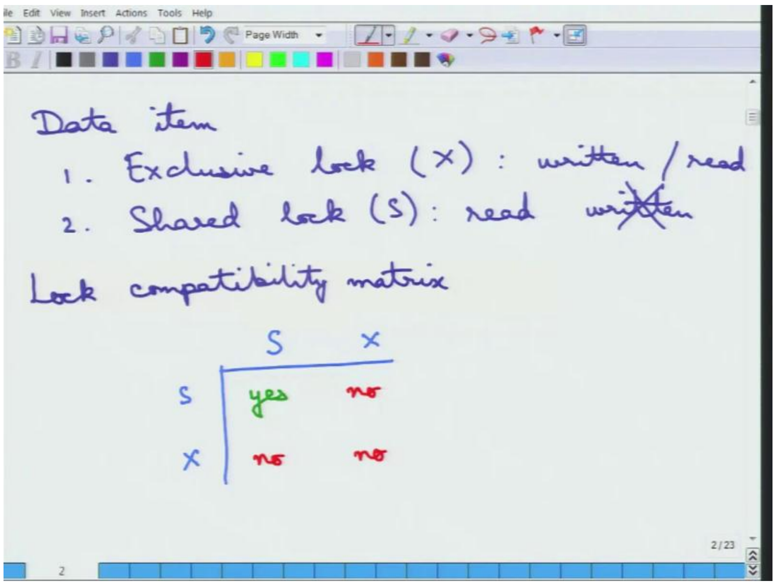
So, this is about concurrency control protocols. So, one of the ways of ensuring concurrency is to the use of locks. We all know that locks are, what ways also use, operating systems also used to ensure concurrency and the databases have no exception, they also use lock. So, locks are what that access the control. So, this control the access to a data item and the requests of lock are essentially merge to, there is a module in the database, which is called the concurrency control manager.

So, this is the module in the database that handles this concurrency control protocols and handles the locks, etcetera. So, all requests for locks are made to this concurrency control manager. The concurrency control manager decides whether to grant the lock or deny the lock, if the grant, if the lock is granted then the transaction essentially occurs that lock and go ahead with that operation on the data item. If it denies, it can actually wait, the transaction generally just waits till it is granted later by the concurrency control manager and then it goes away.

One very important thing is that the operation of locking and data item by a transaction as well as operation of unlocking the data item by a transaction, these two operations must themselves be atomic. So, it cannot happen that two transactions are trying to lock a particular data item. So, only one of them will get the lock or it may happen that none of them will get, that is the separate issue. But, so these are by itself atomic operations, the locking and unlocking, just the operations may themselves the atomic.

So, either a transaction completely gets a lock or it does not get it. So, two transactions cannot get the same lock, even if they I mean request almost in the same time, they will some ways of ensuring that only at most one of the transactions get the lock.

(Refer Slide Time: 02:53)



Data item

1. Exclusive lock (X) : written / read
2. Shared lock (S) : read ~~written~~

Lock compatibility matrix

	S	X
S	yes	no
X	no	no

So, then a data item may be locked, so a data item is what it is locked or unlocked, a data item may be locked in essentially two nodes, the first one is called an exclusive lock. So, this is denoted, so exclusive lock, this is denoted by an X. So, essentially this means that data item can be both written as well as read. So, if a transaction obtains an exclusive lock on a data item, then the transaction can both write and read to the data item. The other type of lock is the shared lock which is denoted by S, which means the transaction if obtain the shared lock on the data item, it can only read.

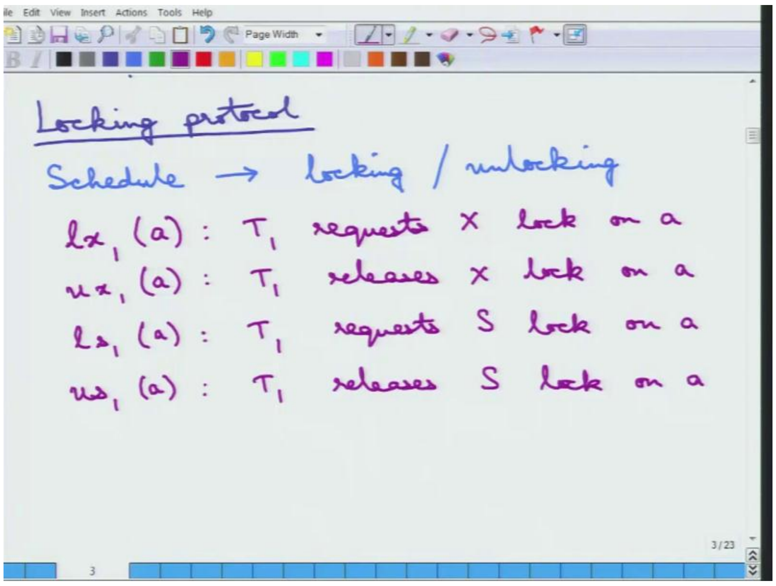
So, write data item cannot be written, so it can be only read by the transaction. So, these are the two important things the exclusive locks and the shared lock and then there is a lock compatibility matrix. So, there is a lock compatibility matrix that essentially defines the compatibility between the different kinds of locks. So, which means that suppose there are two transactions that are also, all locking etcetera we will be assuming on the same data item; otherwise, there is no conflicts and there is no issue.

But, suppose there are two transactions, the row denotes one transaction and the columns denotes another transaction. So, the point is if there is some transaction which has got shared lock and the other transaction wants the shared lock. Can we get it? Yes, it can get it. Suppose, one transaction has got a shared lock and the other wants an exclusive lock, can it get it, no it cannot. So, similarly if one has got the exclusive lock and the other wants the shared lock, it cannot and of course, if one has got the exclusive lock, the other cannot get the exclusive lock.

Now, this is should be very intuitive as to what is happening, this is essentially to get against the conflicts with write. So, if there are two transactions that the both of them want to writes to the same thing, which is both of them are wanting the x lock, that cannot break it. Even if one of them wants read and the other is writing, the other transaction will not be able to get the x the shared lock, because other has got the exclusive lock.

Now, only case where both of the transactions can get is both of them wants to read the data item, in which case there is no conflict. So, both of them can just obtain a shared lock which is why the S to S matrix, the entry for S to S is S and they can just go ahead with doing that. So, the read, read there is no conflict, this lock compatibility matrix does just it codes the conflict that we have already studied. So, that is the thing and as I said earlier, that if a lock cannot be granted in this case, it will be just wait for that. So, using all these locks, we will next move on to the locking protocols.

(Refer Slide Time: 05:43)



Locking protocol

Schedule → locking / unlocking

$lx_1(a)$: T_1 requests X lock on a

$ux_1(a)$: T_1 releases X lock on a

$ls_1(a)$: T_1 requests S lock on a

$us_1(a)$: T_1 releases S lock on a

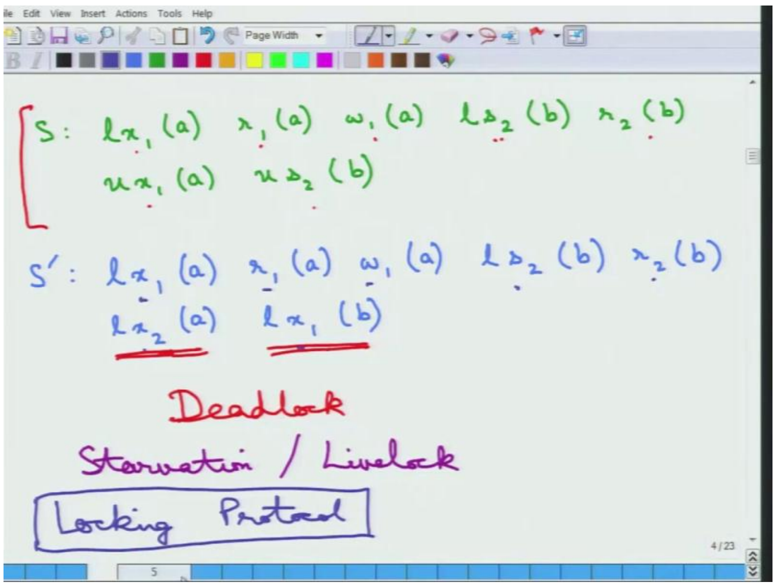
So, the locking protocol is essentially how the transactions must be granted locks, so that the concurrency can be maintained and there is no issue with the correctness. So, the correctness of the schedules be also be maintained. So, we have seen that the schedule we have already seen that the schedule specifies all the read, write and maybe the abort and commit operations as well. In a locking protocol, a schedule must also mention explicitly, all the locking and unlocking operations. So, it must say the transaction one wants a lock, etcetera and so on and so forth.

Now, essentially we cannot guarantee whether the locking or unlocking has been done, because that is done by the concurrency control manager. So, what it essentially tries to do is to request a lock and then unlock that is it. So, now, that is denoted by in the following manner, so when a transaction... So, this is l x of 1 of a transaction, that this notation means that transaction T 1 wants, if transaction T 1 requests an x lock on data item a, so that is the meaning of this, so that is why it is called a lock x.

So, l stands for locking, x means exclusive lock by transaction one on data item s. So, that is the notation l x 1 of a, so if transaction T 1 request x lock on a. Now, when you say u x 1 of a, essentially it means that transaction T 1 releases the x lock on a. So, these are the things, it either requests or release. Then, similarly there is this l s 1 of a, which means transaction 1 requests a shared lock, S lock on the data item a and then, there is an u s of 1 which is transaction 1 releases the S lock on a.

So, these are the four types of operations that will be added to the schedules in addition with the read, write and abort, committed, etcetera to specify the locking and unlocking of this operations. So, for example schedule can be simply specified in the following manner.

(Refer Slide Time: 08:04)



$S: lx_1(a)\ r_1(a)\ w_1(a)\ ls_2(b)\ r_2(b)$
$ux_1(a)\ us_2(b)$

$S': lx_1(a)\ r_1(a)\ w_1(a)\ ls_2(b)\ r_2(b)$
$\underline{lx_2(a)}\ \underline{lx_1(b)}$

Deadlock

Starvation / Livelock

$\boxed{\text{Locking Protocol}}$

Let us just go over briefly what does this mean. So, essentially it means that first the transaction one requests are the exclusive lock on a, once it gets it otherwise it must read once in gets it reads a and then write to a, then transaction 2 request the shared lock on b which will be granted, if there are assuming there are no other transaction since, because these are in the different data item and this is a exclusive lock and this is just shared lock on b which is got nothing to either that is granted then it reads, because it has got a shared lock it can only read it cannot write.

So, it only read which is fine and then this transaction one releases the exclusive lock and transaction 2 releases the shared lock and that is the end of the schedule. So, that is how are schedule will be specified in this. Now, let us see an example of what needs to be, so this is all fine as a schedule. Now, let us consider another schedule and see what may happen which such a schedule, now note very importantly what we are doing here is that the following.

So, transaction 1 once an exclusive lock on a assuming there are no other transactions and assuming there is nothing else going on, this lock will be granted, because there is no conflict nothing. So, transaction one gets that lock only then it reads and then writes, transaction 2 request the shared lock on b again that will be granted, because there is no problem then it reads. Now, what happens is that transaction 2 once an exclusive lock on a and transaction 1 once an exclusive lock on b.

Now, the important thing comes is that transaction 1 is already holding and exclusive lock on a. So, unless transaction 1 releases it or unlocks it transaction 2 cannot get another exclusive log on a, because that is what it is not allowed by the lock compatibility metrics. So, it will not be getting, so it will keep on waiting, so transaction 2 at this point will keep on waiting for the exclusive lock. On the other hand transaction 1 once an exclusive lock on b, now transaction 2 holds the shared lock on b it has not released it.

So, transaction 1 wants an exclusive lock again this is not going to be allowed and this will keep on waiting. So, now, essentially what is happening is here is very, very important. Transaction 2 keeps waiting for an exclusive lock on a which it is not going to get unless transaction 1 releases, transaction 1 on the other hand is waiting for a exclusive lock on b, which is again not going to be granted unless transaction 2 releases.

Now, you see this is the classic dead lock situation, transaction 1 once a lock which it is not going to get and unless it is I mean and it is not releasing other locks and transaction 2 once another lock which is again it is not going to get. So, it is not going to release other lock, so both the transactions are just waiting for locks and for the other transaction 2 and none of them is going to release it, so it is the classic dead lock situation.

So, the concurrency control protocols whatever this protocol in the schedule etcetera, the locking unlocking may resulting the dead lock I mean it is only says when to request for the lock it does not mean that the lock is got immediately, it only request for a lock and depending on the concurrency control manager it may or may not get it and it is especially something is that it can never violet the lock compatibility matrix. So, if there is another transaction that holds and exclusive lock, it cannot get another exclusive lock and shared lock on it.

And if another transaction holds the shared lock, it cannot gets on exclusive lock that is the lock compatibility matrix. So, it may resulting dead locks, now otherwise what will happen is that there may be a violation of the correctness. So, the concurrency control manager will never sacrifices on the correctness it will not grant this lock. So, what may happen is that they maybe the situation of dead lock that is the one thing, the other thing is starvation a live lock that is another phenomenon that may also happen.

So, what is essentially it is the phenomenon of starvation or live lock is that suppose there are many transactions and transaction 1 holds the lock on some item a. Now, both transaction 2 and transaction 3 once it as it happens that once transaction 1 releases it, transaction 3 crabs the lock, so transaction 2 does in it. Now, before transaction 3 releases it, transaction 4 again wants the same lock on the same data item.

So, once transaction 3 releases transaction 4 crabs it and transaction 2 again does not get it and then again before transaction 4 releases, transaction 5 wants it, so transaction again does not get it. So, essentially there is no dead lock in the system, in the sense that the transactions are proceeding some of the transactions are at least proceeding is not that all the system is altered, but transaction 2 is starve, it is starve.

Why it is called starvation? Because, it is not getting the lock that it once some other transactions are getting it. So, that is the phenomenon of starvation and live lock and that can also happen if this schedule is the just requesting for a locks and releasing it that can also happen. So, that is the point of schedule, so then what it is being done is that now what we are going to next study is this, what is called the locking protocol.

So, essentially the locking protocol will try to specify how a transaction should request for locks and how it should release the locks, such that the correctness is not violated. So, that is the locking protocol that we are going to study next.