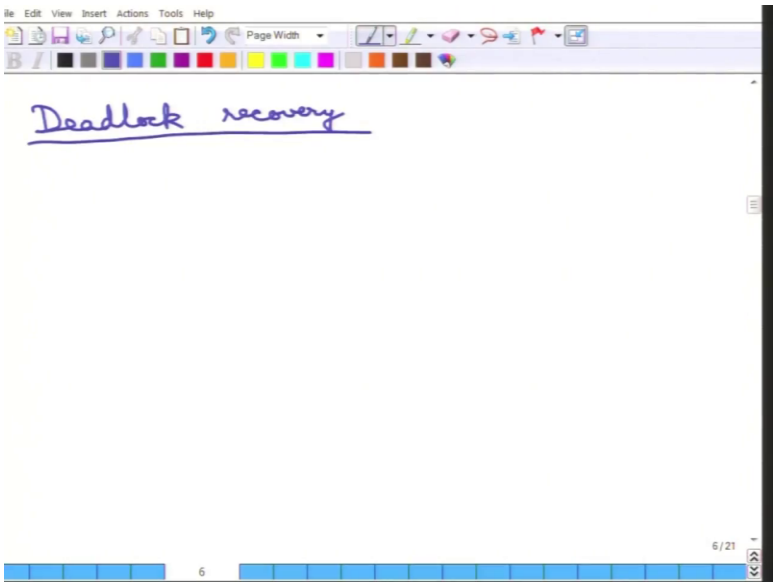
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

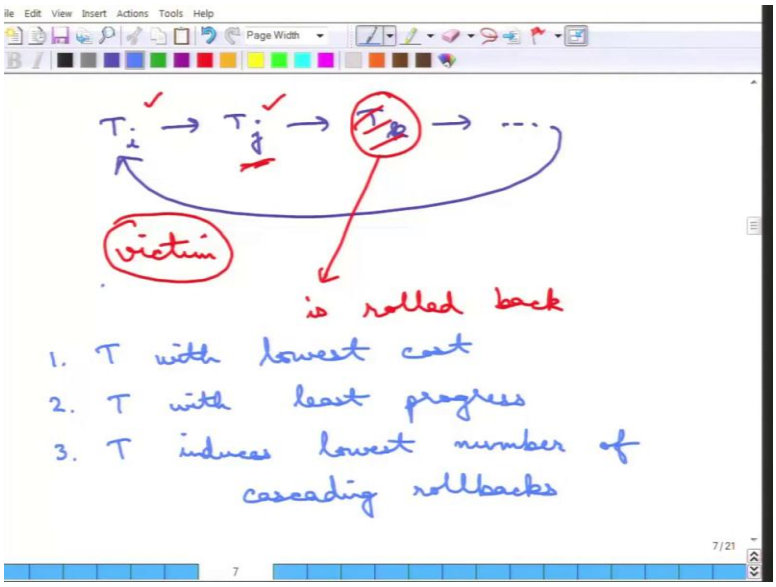
Lecture - 44
Concurrency Control: Deadlock Recovery and Update Operations

(Refer Slide Time: 00:09)



The question now is, suppose the cycle has been detected and it now needs to be recovered.

(Refer Slide Time: 00:17)



So, how does one recover; that means, so here is the idea. So, cycle meaning T i goes to T j, T i is waiting for T j, this T i, T j waits for T k and this dot, dot, dot and finally, that is waiting for T i. Now, to break the deadlock one of them must be aborted, one of these transactions must be aborted, so one of them must be made a victim. So, suppose arbitrarily T k is made the victim; that means, T k is now rolled back, so all the resources that T k holds is released, which means T j can now pass; that means, T j will finally, finish and; that means, T i can then pass and so on and so forth, so this is the whole effect of this.

So, one of them must be chosen as the victim of this. Now the question is, which transaction is the victim? Now, there are many schemes to do that, but there are essentially some heuristics to this. So, the first one is the transaction that has the lowest cost is chosen as the victim. Now, what does that mean? What does it mean to say the lowest cost? Remember, that these transactions are essentially operations, the read, write of those kind of operations and each such read, write operations will have some cost, this is by the query processing engine, they will estimate some cost.

So, the transaction that has the lowest cost if that is roll back; that means, that the least amount of work is supposed to be read. So, it essentially is that, it can finish off the most quickly, so if that is read that is not a problem, so that is the one with the lowest cost; that is one heuristic. The other heuristic is that the transaction with the least progress. Now, what does that mean is that even though, we were in the first heuristic, even though the transaction with the lowest cost is being chosen, that may not be a good one, because even suppose the, it has the lowest cost, but it has almost completed.

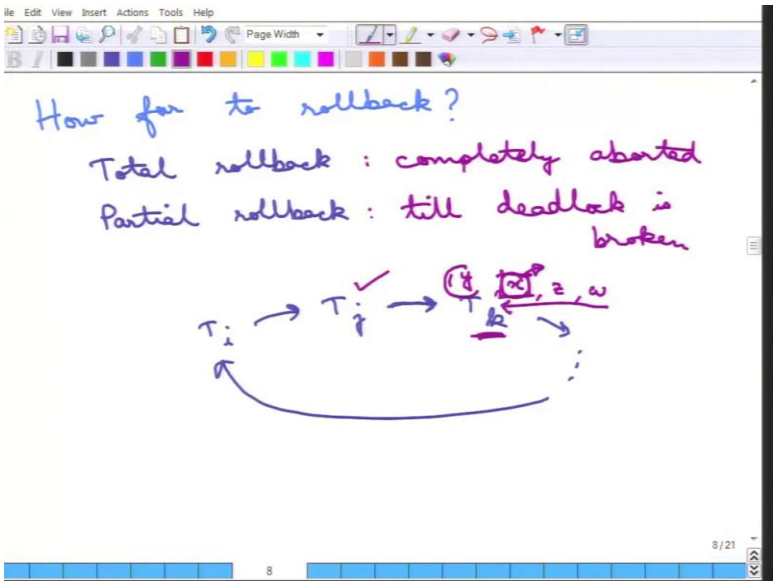
Now, if that is rolled back, then all that work needs to be done. In the second heuristic what is done is that, the transaction with the lowest progress is being chosen, lowest progress meaning, it has gone into the order of operations into the transaction the least amount. So, it has essentially done the least amount of work, so it has essentially completed the least amount of work. So, if that is being rolled back or that is being made the victim, then the least amount of work is being lost.

So, that is the rational of the second heuristic and the third heuristic is the one that induces the lowest number of cascading roll backs. Now, this is should be very, very intuitive and this is very, very important to understand, that what happens in the case of a

database is that if a transaction rolls back, it may have a series of cascading roll backs. So, even though the transaction may have the greatest progress or greatest cost, etcetera or if, so and if it has being and it has gone through much far etcetera and it has or the other way around it has not gone through much far etcetera, but if it rolls back, then it will cause a lot of cascading roll backs.

So, essentially all those cascading roll backs meaning all the transactions that are now roll back as the effect of this roll back will need to be redone, so that is our big cost. So, essentially the transaction, which has minimal effect on the other transaction in terms of cascading roll backs is chosen. So, these are the three heuristics of how to choose the victim or how to choose the transaction to roll back.

(Refer Slide Time: 04:01)



The next question comes is how far to roll back, so the earlier question was, which transaction to roll back and then, the question is how far to roll back. Now, of course, intuitively one may say that, if a transaction is rolling back it must roll back to the complete beginning of it, the complete start of this, so that is called a total roll back. So, total roll back means the transaction, essentially all the operations done by the transaction are undone and the transaction essentially just restarts, it is completely aborted.

So, this is essentially, it saying that completely aborted, the transaction is completely aborted. What can also be done is something called a partial roll back can be done. Now,

what is the idea of a partial roll back is that, may be not all the operations in the transaction are wrong, may be not all the operations in the transaction have caused the problem. So, it is rolled back, so only that for which there is a problem. The problem is in the sense of… Remember, that we are not talking in the problem, in the sense of recoverability or etcetera, we are talking in the sense of deadlock.

So, roll back till deadlock is broken, now that is the interesting thing to do now. Now, let us just go back to the example that we did earlier, to understand what is being happened. So, suppose T i waits on T j that waits on T k and that waits on so on and so forth, then this go comes back.

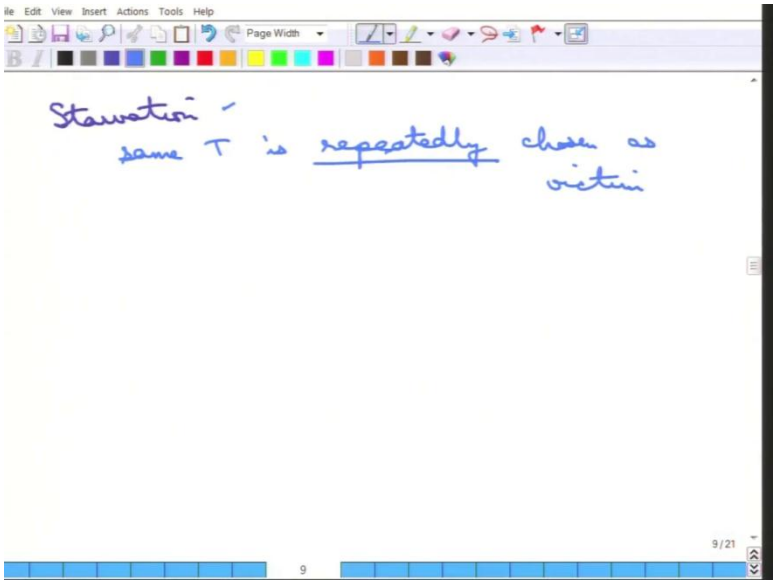
Now, suppose this T i waits for T j is being done, because of a data item x. Now, before that suppose T k has held the data item y and suppose after that it held's the data item z, then it holds the data item w and so on and so forth.

Now, the problem is with this x, so T k is a victim, now if x is released, then T j can succeed and then, once T j succeeds T i succeeds and the deadlock is broken. So, the point is it will be a roll back only till the point of the problematic data item due to, which T j is rolled due to which T j is waiting. So, why this operations that involve y, the data item y did not be rolled back? Because, y is not causing any problem, so it rolls back only to the point, where there is a problem and then it is released.

But, this requires a lot of schematic analysis and the thing is that, it has to be known that, which is the data item it can hold, which is this waiting for which is of course known, but it must be also known that this partial roll backs can be problematic. Because, even though it can be said that all the operations of y may not be rolled back, but there may be some operations of y, that affect the later operations of z and w and so on and so forth, so all of these must be taken care off.

In general, by keeping a count of which operation comes after which operation, which the transaction they anyway keep at it, then it can be rolled back till the point, where the deadlock is broken, so that is the thing about how far to roll back.

(Refer Slide Time: 07:03)

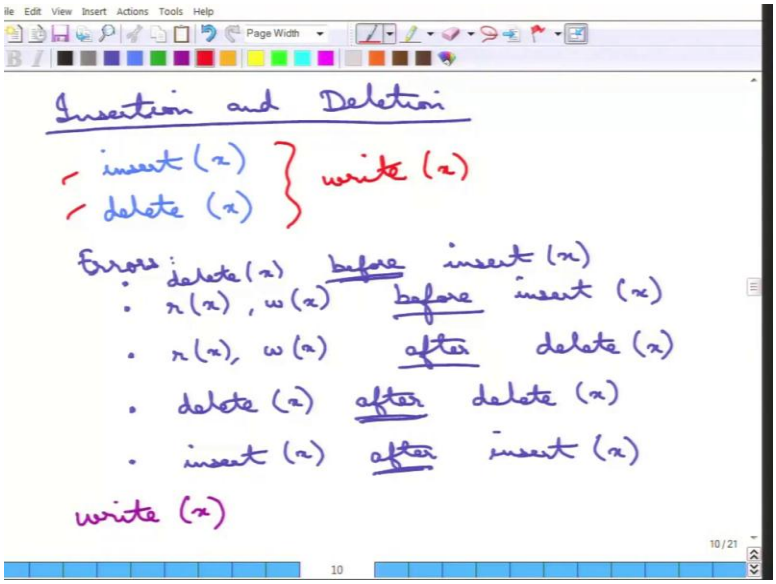


Then, the question is the other thing is that starvation in this deadlock recovery schemes, the starvation may happen, because the same transaction is repeatedly chosen as the victim. So, that is why this starvation may happen. So, repeatedly chosen as victim, this may happen, because this transaction is doing the least progress or transaction has a lowest cost, etcetera, this is repeatedly chosen as the victim, so start starvation, so it may go into this thing.

So, one interesting way or one way of breaking this problem of starvation is that when a transaction is chosen as a victim, a count is kept. So, a victim count is kept for each transaction and to break the starvation problem, there may be a threshold number of times, when a transaction is a victim, so suppose it is three. So, once a transaction is chosen victim for three times, it will not be chosen as a victim for the next time.

So, it can be starved for three times, but not any further, so that is to just break the thing about starvation. So, that is this all the things about this locks and the deadlock recovery, etcetera.

(Refer Slide Time: 08:16)



Now, there is one thing that is left for the concurrency control protocols is we have been talking in the concurrency control protocols all along about read and write. Now, there are two more important operations in databases that are quite frequent actually in transactions, that we have not talked about at all, these are insertion and deletion. So, what happens when a new tuple is inserted into a table or when a tuple is deleted from a table or some attributes are updated, etcetera? Remember, that updates can be handled always as a deletion and then, an insertion.

So, what we will talk about here is the insertion and deletion, so insertion and deletion, so what happens in the case of insertion and deletion. So, for a data item x there are two operations, that we need to worry about, which is the insert x and a delete x. So, insert x meaning this x is newly inserted into whatever place it holds to and delete x is from ((Refer Time: 09:18)). So, let us summarize the logical errors is that, so the errors that can happen due to insertion and deletion the logical errors that can happen is that, there can be a read of x or write of x before insert has taken place.

So, one transaction wants to read to an x or write to an x, but which has the other transaction, which is supposed to insert it has not yet done, so that is of course, an error. Then, the read x and write x may be happening after the delete of x, so one transaction to suppose to read or write this was late and the another transaction came and by the time

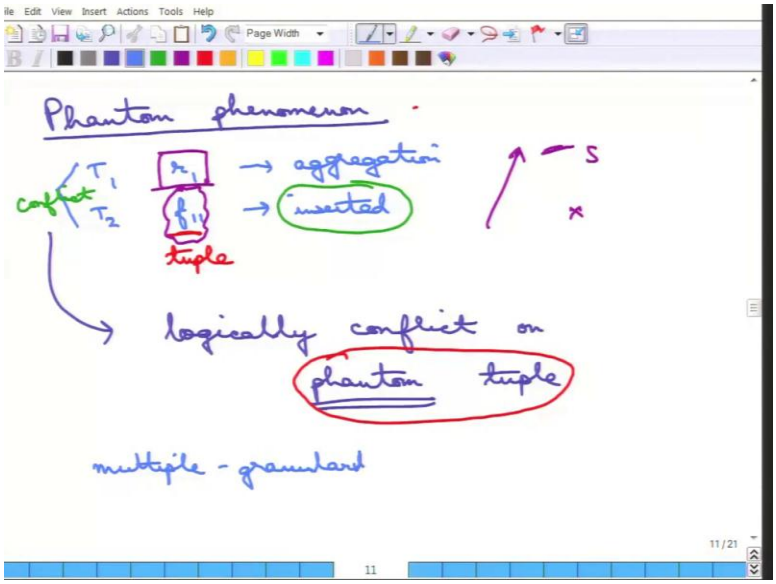
this read and writes came for this transaction another transaction came and deleted it, so that is also needed.

Then, there are other kinds of errors, which are happening, because these are not idiom put in operation. So, once the data item is deleted it cannot be deleted again, so the deletion operation is not idiom putted; that is error actually the same for insertion. So, once a data item is inserted another transaction cannot issue another insertion operation, because it is already in the place already in the record or database etcetera, so these are the problems with insertion and deletions.

Now, note that everything can be very simply handled by assuming that read insert x and delete x by assuming that insert x and delete x at similar to write x. So, all the conflict that happen with write is supposed to be happening with insert and delete; that means, any insertion is a conflict with write any insertion is a conflict with another insertion any insertion is a conflict with the delete. So, one thing that I have left out is that delete x is a problem before insert x, so once something is supposed to delete it, but it did it before, so that is a problem as well.

So, if insertion and deletion are treated like write x and all the conflicts, that write x has is propagated to insertion and deletion and then, the lock compatibility matrix and then, the conflicts and everything is handled in the same manner as the write x, then everything will be correct. So, that is the case with insertion and deletion. So, that is the way to handle insertion and deletion.

(Refer Slide Time: 11:41)



To finally, finish the concurrency control protocol we will talk about one interesting phenomenon, which is called a phantom phenomenon, so a phantom phenomenon is the following is that. Suppose there is a transaction T 1; that is reading a relation r 1 for, let us say competing an aggregation or something like that ((Refer Time: 12:02)), so this is the this thing.

And then, in the meanwhile there is a transaction T 2, which comes and updates some field units some in, so this is inserted. So, this some field is inserted or some tuple is inserted and so on, so forth into the thing. So, then T 1 and T 2 of course, as we can see this is a conflict this is of course, a conflict, because the aggregation will be wrong if it is allowed to insert. But, note that strangely enough this is hard to detect, because where is the insertion happening the insertion is not happening at the relation r 1 the insertion is happening inside a tuple or inside a field or this is a new tuple; that is being inserted.

So, this is a new tuple, so just to highlight this is a tuple, so this is a new tuple; that is being inserted, that tuple of course, must not locked by T 1 was there was no tuple earlier. So, when r 1 is locked even in the is mode and all of these things everything that is locked below cannot lock this f 1,1, because f 1,1 does not exists at all this was this tuple did not exists, so this is being inserted.

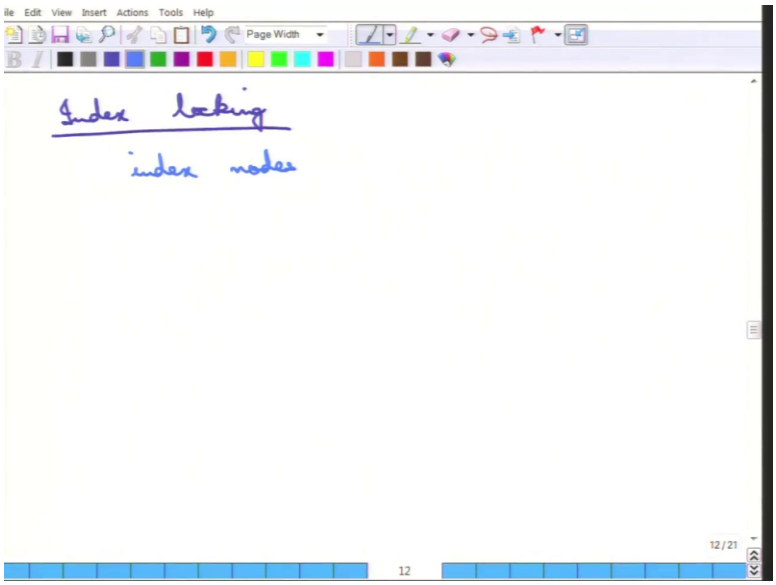
So, how to handle this is why it is called a phantom phenomenon, because when r 1 is being locked they do not lock this, but, they logically conflict on something called a

phantom tuple, so this is logically logically conflict on a phantom tuple. Why is this phantom? Because r 1 does not know the existence of it, so for r 1 this f 1,1 is a phantom tuple f 1,1 does not to be in exist for transaction one, so for r 1 with for transaction.

So, this is a phantom tuple; that is why this is one of the hardest problems in databases to catch. So, the insertion one of thing is hard actually this phantom phenomenon, but, so this is known as a phantom tuple this f 1,1 and this whole phenomenon this is known as the phantom phenomenon. So, that is the thing of this, so exclusive lock know how to solve this essentially is that one must get an exclusive lock on this relation r. So, this is that exclusive lock; that is being brought, so whenever this is being read etcetera this exclusive lock.

So, the multi granularity locking actually can solve it, whenever this thing is trying to insert it checks all the way up its parents and see that r 1 is locked in as mode, because this is being done, so this cannot be inserted. So, insertion essentially an x lock, but this is already locked in a shared mode, so it cannot get an exclusive lock. So, the multiple granularity locking actually solves it, so the solution is a multiple granularity locking that we saw earlier, but this is just to highlight that this is one of the very hard problems, so interesting problems to catch, so this is the phantom phenomenon.

(Refer Slide Time: 15:10)



And just one thing to end the concurrency control protocol, we have been talking about the databases relation etcetera and all of these things, but the databases also use the index

structures, so the index locking must also be taken care of. So, the varieties of things in the index must be locked. So, the index nodes may need to be locked and the index nodes locking the index nodes essentially meaning locking the data items that it the index node corresponds to etcetera.

So, this may also helps to avoid the phantom phenomenon, because the insertion etcetera is always happening through the index lock and the index through the index node and the index is already locked, because it will touch that particular tuple. So, that finishes the module on concurrency control we will next go over no SQL and big data.