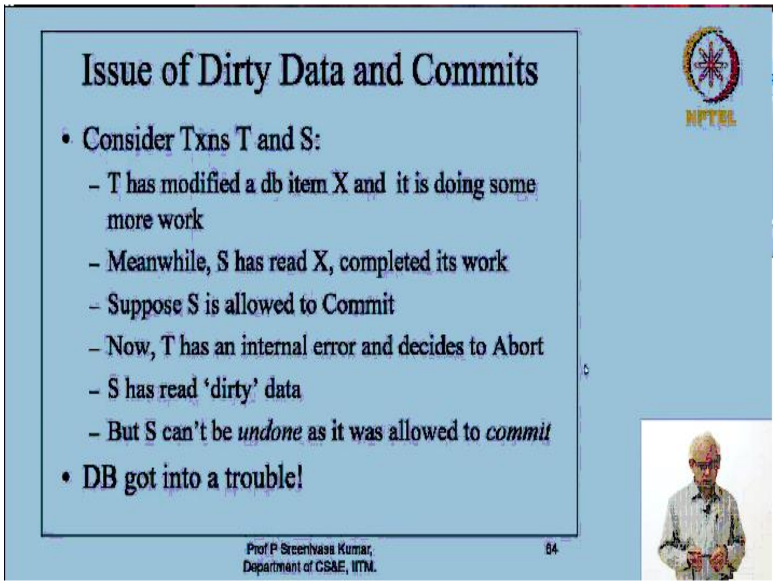
Database Systems
Prof. Dr. Sreenivasa kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 41
Recoverable Schedules and Transaction Isolation Levels

(Refer Slide Time: 00:27)



Issue of Dirty Data and Commits

- Consider Txns T and S:
  - T has modified a db item X and it is doing some more work
  - Meanwhile, S has read X, completed its work
  - Suppose S is allowed to Commit
  - Now, T has an internal error and decides to Abort
  - S has read 'dirty' data
  - But S can't be *undone* as it was allowed to *commit*
- DB got into a trouble!

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 84

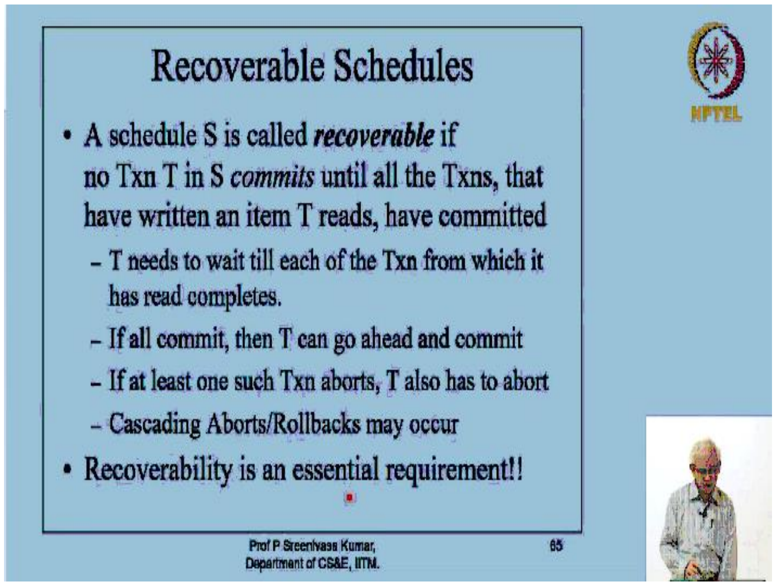
So today, we will spend a short time and trying to wind up the course. So, let me because I have a few points to discuss today. So let me sort of repeat a few things from the last lecture. We were discussing failures, system failures, and then looked at the various solution options that we have the differentiable tool that is used with this log and various kinds of logs. And, depending on the log, what should be the recovery procedure, all those issues we have discussed.

Now an important aspect is an aspect called recoverability, which I briefly I have introduced in the last lecture also. So, the whole thing arises from the fact that you know, if transactions commit in certain kind of order, then it is possible for us to get into for the database to get into trouble. So, I have given a sequence of you know, events here that a transaction T has modified a db item and it is doing some more work.

Meanwhile, some so this is the case of the reading dirty data. So, meanwhile some other transaction S has read that item X completed it is work, but suppose S is allowed to commit then in such a scenario, it is very clear that we will get into trouble because we really do not know what T is going to do in the future. So, T might finish successfully in which case we may not have much trouble.

But if we for some reason, T you know, for example, having encountered an internal error, it might decide to abort in which case it would turn out that S has actually read dirty data. So, the whole thing has come about because we have allowed S to read an item that has been modified by a running transaction and then instead of waiting for the running transaction to come to a conclusion, it has this transaction has gone about and then committed and we have allowed such a commit to take place. So this is the classic scenario in which we will get into trouble.

(Refer Slide Time: 02:59)



Recoverable Schedules

- A schedule S is called ***recoverable*** if no Txn T in S *commits* until all the Txns, that have written an item T reads, have committed
  - T needs to wait till each of the Txn from which it has read completes.
  - If all commit, then T can go ahead and commit
  - If at least one such Txn aborts, T also has to abort
  - Cascading Aborts/Rollbacks may occur
- Recoverability is an essential requirement!!

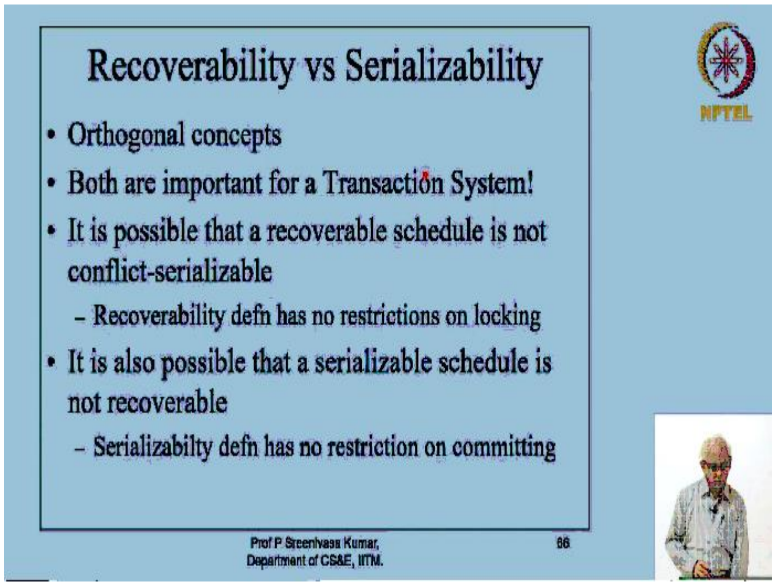
Prof P Sreenivasa Kumar, Department of CS&E, IITM. 65

And so we isolated this and then defined what are called recoverable schedules. So a schedule S, this is not a transaction or this schedule. So it is a sequence of operations of several transactions. So, the transaction is the schedule S called recoverable if in that schedule no transaction commits until all the transactions from which it has read have actually committed. So, basically, this transaction T needs to wait till each of those transactions from much it has read some item completes. So the completing could be that they commit or they may even abort.

So if all of them commit, then T can go ahead and then ask for a commit because the relationships work more. But if any one of them aborts, then it is necessary that T also has to abort because it has dirty data. So, if in such a scenario we also encounter this phenomenon called the cascading aborts or cascading rollbacks because if this transaction T this has to abort then all those transactions which have read an item that has been released by transaction T also have to unroll.

And because they have to unroll from where has read an item that has multiplied by the amount of cascades. So, cascading rollbacks may occur. But then we thought this property being satisfied by schedules because of the database would not be able to give durability assurance at all right. So, so, what durable requires is that if some transaction commits, then the effects of that particular transaction are permanently recorded in the database on the disk, such a kind of semantics will not be able to provide, if we do not take care about observing the securability constraint.

(Refer Slide Time: 05:24)



Recoverability vs Serializability

- Orthogonal concepts
- Both are important for a Transaction System!
- It is possible that a recoverable schedule is not conflict-serializable
  - Recoverability defn has no restrictions on locking
- It is also possible that a serializable schedule is not recoverable
  - Serializabilty defn has no restriction on committing

Prof P Sreenivasa Kumar, Department of CS&E, IITM.

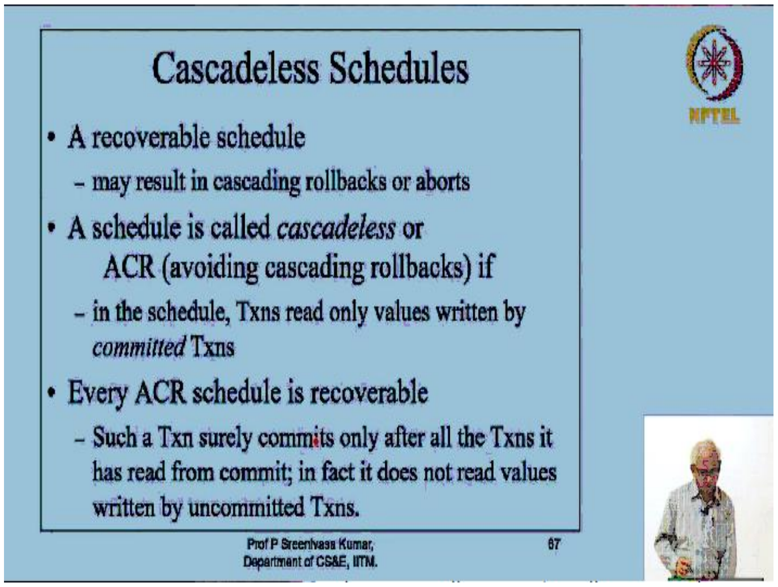
66

Now, we have also discussed this concept of serializability where this has more to do with how do transactions operations interleave and what kind of transaction operation interleavings are acceptable or desirable. So, in that context, we have discussed the issues and then we have identified the notion of serializability and one particular way of achieving that serializability called conflict serializability which is I mean achieved through lock based protocol.

So, we are discussed two phase locking protocol and if transactions follow two phase locking protocol then serializability is guaranteed. But then these we realize that serializability does not, you know, put any restrictions on the order in which commits have to happened things like that. So, strict serializability definition does not include anything about commits, whereas, and also recoverability has nothing do about interleaving of operations and things.

So, these 2 things are actually orthogonal definitions, but then in practice what we need actually, we need schedules that are both recoverable as well as serializable. Only then that particular transaction system will be able to achieve all the desirable properties called the especially with all the ACID properties comes isolation and recoverable. Consistency of coefficient where programmers responsible.

(Refer Slide Time: 07:13)



Cascadeless Schedules

- A recoverable schedule
  - may result in cascading rollbacks or aborts
- A schedule is called *cascadeless* or ACR (avoiding cascading rollbacks) if
  - in the schedule, Txns read only values written by *committed* Txns
- Every ACR schedule is recoverable
  - Such a Txn surely commits only after all the Txns it has read from commit; in fact it does not read values written by uncommitted Txns.

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 67

So, now, in the context of this entire thing, we need to also think about whether we can avoid these cascading rollbacks. Can we avoid cascading rollbacks? Because rollback is a costly operation, we will have to undo a lot of efforts that have been done on the database and all that. So, in such context we can decide a schedule is so we introduced this notion called cascadeless schedules, this cascadeless schedules or those kind of schedules in which we do not have this need for cascading rollbacks.

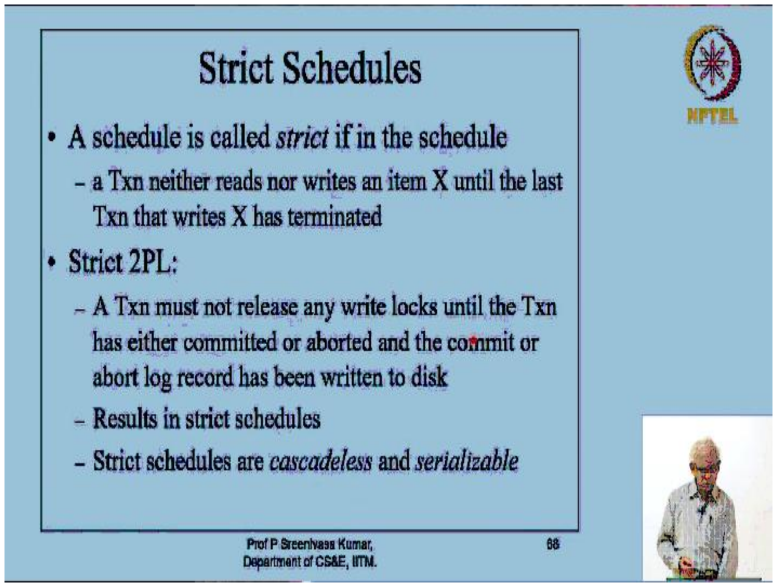
So, basically such as schedule is called ACR schedule also which basically stands for avoiding cascading rollbacks. ACR basically says avoiding cascading rollbacks, so schedule is called cascadeless or ACR schedule in this schedule transactions read values written by committed transactions we can reading some item I must ensured that is in last written by our overload that last is a committed but how do I know as a transaction I will not know.

So, we have to cleverly again modify our locking protocol in such a way that when transactions are running, they will not be allowed to read items that have not been tempted by that have been written by uncommitted transactions that have been. So this will shortly see how we can modify the two phase locking protocol to achieve that. But once this is there, it is very clear to you know that every one of these avoiding ACR schedule is indeed recoverable, because so we have put this restriction saying that it has read from committed transaction.

So such a, so a scheduled in this schedule, every transaction, surely, you know will commit only after all the transactions that have that it has read from commit and that is what the recoverability constraint this, if you are committing, then if you read some item that that was modified by some other transaction, you should wait for that to commit before you commit, that is what the recoverability constraint.

But now, if you do not even read the items, unless that fellow has committed, then you are guaranteed that you are only touching, picking up items that have been written by committed transactions and so you will surely commit only after All of them commit. So, any such schedule in which this property can be guaranteed somehow the transactions read only values written by committed transactions will be cascadeless and such a thing is actually not very difficult to achieve because all that you do like this locking schedule is in your hands So, now you dictate as to how are the 2PL has to be appropriately changed.

(Refer Slide Time: 10:40)


Strict Schedules

- A schedule is called *strict* if in the schedule
  - a Txn neither reads nor writes an item X until the last Txn that writes X has terminated
- Strict 2PL:
  - A Txn must not release any write locks until the Txn has either committed or aborted and the commit or abort log record has been written to disk
  - Results in strict schedules
  - Strict schedules are *cascadeless* and *serializable*

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 66

So, for locking based methods for concurrency control. Notice that there are other ways of achieving concurrency control especially we have we did not have time to discuss this other matter. They are called timestamp based approaches for concurrency control these timestamp based concurrency control are actually also called optimistic concurrency control techniques in since that they will let the transactions run.

And then check whether everything is proper and if not so take some corrective action. So assuming that most transactions do not get into double digits, so there is an optimistic way of doing that. Whereas the locking based concurrency control based on two phase locking protocol is a pessimistic approach it will assume that if you do not do anything, transactions will conflict with each other and so if it is enforcing conflicts serializability and thus making sure that the schedules that araised are conflict equivalent to some serial schedule serial schedules are safe.

So that is the equation. So, when we restrict ourselves to locking based concurrency control methods, a way of achieving this cascadeless schedules is to modify the two phase locking protocol and make it into a strict two phase locking protocol where we will impose that a transaction must not release any locks especially write locks. Write locks are those which are acquired for the purpose of writing updating an item.

So, any write locks until the transaction has either committed or aborted and the commit or abort log record has actually been written. So till that time you do not release locks. So if you do not release locks if a transaction does not release locks, then there is no chance for any other transaction to acquire a lock on some item and then used. So, in this case, you are keeping all the locks with you till you commit or know about.

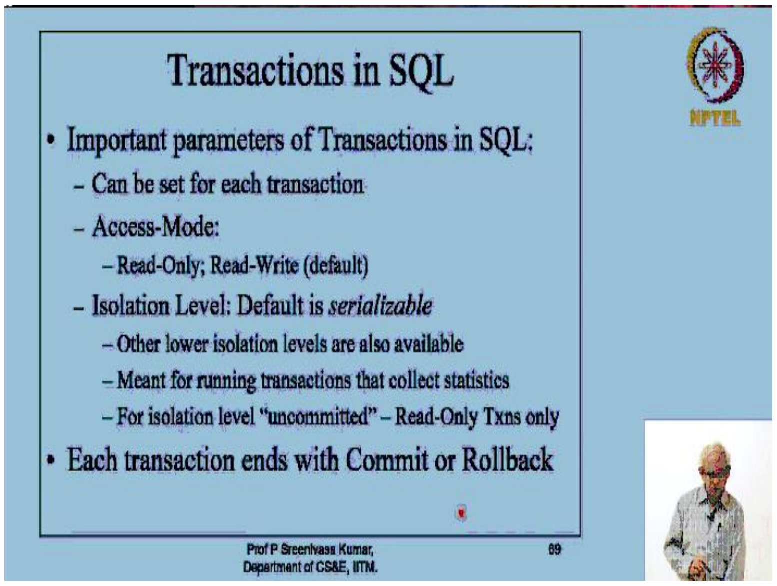
So if you commit, and then you release items, then anybody who else, anybody else who you reading these items that you have modified, are actually reading items that have been written by a committed. So there is no issue, so cascadeless schedule levels, in case this transaction has aborted, then obviously, those transactions have to be aborted. But this aborting a transaction is not a major issue.

We can do that just like we have done error recovery procedures, we can take one particular transaction undo it is operation. So this modification results in what are called strict schedules. So strict schedules are those schedules in which the transaction neither reads nor writes any item until the last transaction that wrote X has been terminated. We do not know that are successfully or otherwise terminated. So this will result in what are called strict schedules.

And we can so strict schedules are following 2PL is a strict 2PL that is more stricter version of two phase locking, why it is a stricter version of two phase locking? Whereas two phase locking, all that we said is that, do not ask for a lock. The moment you start issuing unlocks. The moment issue if you the moment you start releasing locks do not ask for any more lock that is all we said there, but here we are saying that keep all the locks if you and then release them at the end.

So there is no question of asking for any more locks you kidding me locking phase and then at one go you release. So these are two phase locking protocols and so they are serializable they will result in serializable schedules and they are also cascadeless schedules because they do not allow any transaction to be an item that has been uncovered. So these are this so strict two phase locking is very popular protocol that is used in most of the relational database systems and it touches cascadeless serializable values and that is how locking systems prompted.

(Refer Slide Time: 15:59)



Transactions in SQL

- Important parameters of Transactions in SQL:
  - Can be set for each transaction
  - Access-Mode:
    - Read-Only; Read-Write (default)
  - Isolation Level: Default is *serializable*
    - Other lower isolation levels are also available
    - Meant for running transactions that collect statistics
    - For isolation level "uncommitted" – Read-Only Txns only
- Each transaction ends with Commit or Rollback

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

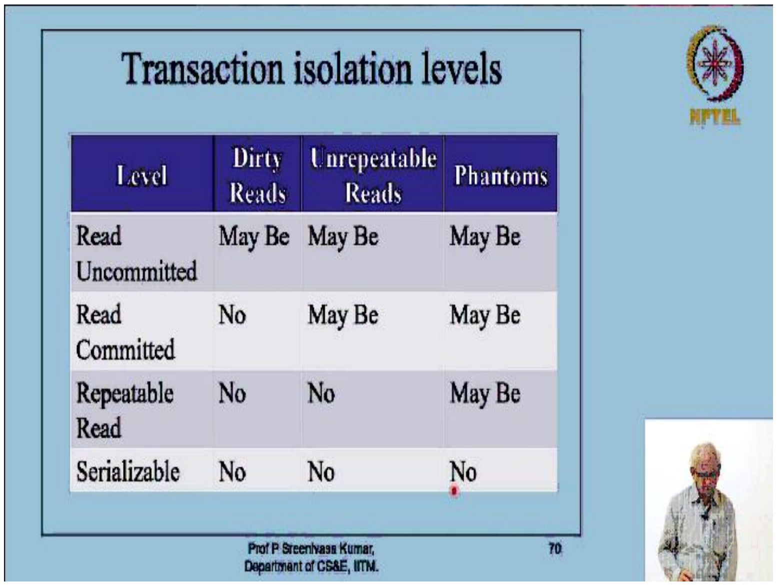
69

So now let us briefly look at the issues how transactions in SQL work. So, there are a couple of parameters for the transactions in SQL that you can set up one is called the access mode, so it called access mode, the other one is called isolation level. So, there are 2 access modes; read only access mode and read write access mode. The default is read write access mode and isolation level is something you know. So, in practice it is of course, a theoretically all the schedules have to be serializable schedules.

Then we are guaranteed that the transaction system is running perfectly well. But in practice, SQL would like to give the users an option of running their transactions in a slightly relaxing isolation level, the strictest isolation level is serializability, so the default is serializablilty if you do not say anything it will run in serializability isolation level. So lower isolation levels are available and you choose it at your own wish.

So, actually these are meant for running transactions that kind of collect statistics and aggregates things like that where you are not really so bothered about the up to date values and things that may not matter to you when you are simply completing some averages. And this one particular isolation level called uncommitted. It is actually dangerous. So, SQL will take an extra precaution to allow only read only transactions to run such at a lower isolation level. So, they will actually you know pick up some dirty data to reading them, but they will not be allowed to modify anything in the database. So, each transaction ends with a either commit or rollback.

(Refer Slide Time: 18:23)



Transcation isolation levels

Level	Dirty Reads	Unrepeatable Reads	Phantoms
Read Uncommitted	May Be	May Be	May Be
Read Committed	No	May Be	May Be
Repeatable Read	No	No	May Be
Serializable	No	No	No

Prof P Sreenivasa Kumar, Department of CS&E, IITM.

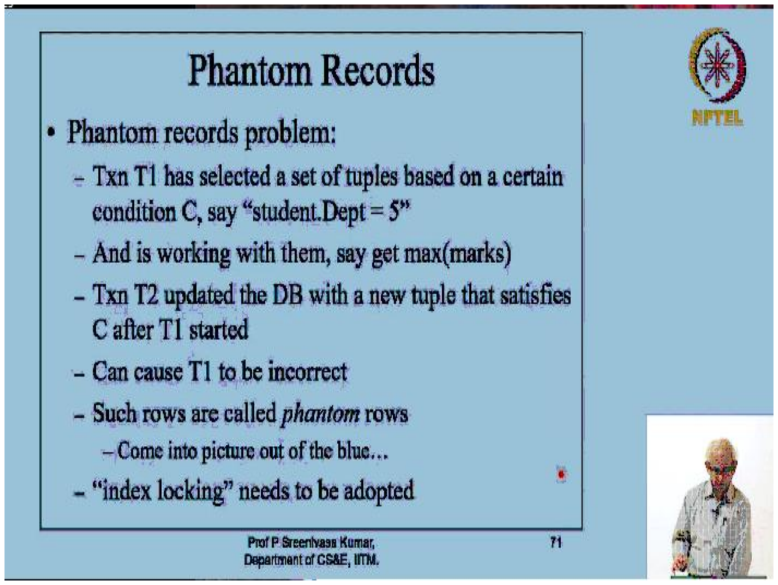
70

So, these are the isolation levels, the transaction isolation levels. So, the highest isolation level is the serializability schedules. So, in that so these are the various kinds of problems that might arise if you choose any lower level of isolation. So, there are what are called read uncommitted. So, this is the lowest level of isolation that one can use but this is the most dangerous level. So, SQL allows you only to run read only transactions in this mode, so, access mode will be automatically set to read only if you choose to say.

So all these problems might arise that is reads might arise, unrepeatable reads might arise and also what are called phantoms arise shortly talk a little bit about this phantoms, then the next highest level is read a higher level is read committed, where dirty reads do not occur, but unrepeatable reads might occur. We have discussed this on unrepeatable reads and phantoms also as repeatable reads, no dirty data will be read and know, all the reads are repeatable.

But phantoms might arise I will tell you what was phantoms are and we of course the best thing to do is to run your transaction are serializable in which you are guaranteed that none of this problem arise.

(Refer Slide Time: 20:04)



Phantom Records

- Phantom records problem:
  - Txn T1 has selected a set of tuples based on a certain condition C, say "student.Dept = 5"
  - And is working with them, say get max(marks)
  - Txn T2 updated the DB with a new tuple that satisfies C after T1 started
  - Can cause T1 to be incorrect
  - Such rows are called *phantom* rows
    - Come into picture out of the blue…
  - "index locking" needs to be adopted

Prof P Sreenivasa Kumar, Department of CS&E, IITM. 71

So, the problem phantom is something which is like this. This is to do with so supposing some transaction, you know has selected some tuples based on certain condition, say for example, student department equals 5. So a bunch of tuple says it has locked all of them and it is going ahead and working with them say getting max marks. So, a transaction T2 will come into picture update the database with a new tuple that satisfies this condition.

So, this record actually was not even existing in the database when the T1 started. So it has started and it has locked certain items which it is going to use, so obviously, if T2 enters some record, it might affect what is being done by T1 suppose it is computing max marks including that. So, this kind of rows which get into the picture after you know, the transaction actually started running are really are called phantom rows because they kind of come into the picture.

So, these things cannot be interdicted by any of these things you have to take some extra precautions. So the usual locking protocols will not so you have locked whatever you wanted to do but so nobody will be able to touch it but then here is a new thing that you should have locked but it was not existing at the time. So, this is a phantoms. So, there is a solution for this and that we will not go into a lock details about that but it is called index locking.

So, it is also there is a valid thing called predicate locking. So basically you should not just lock blocks or you know database items, but you must also put somewhere a record saying that anybody who satisfies this particular condition has to be locked, it should not be entered into this lock. So that is what is called index locking. So that is a very advanced concept. So, using such techniques, they will be able to solve this issue of phantoms in such context.

So with that, so choose the isolation levels as serializable but of course, such some statistics collecting transactions which you do not mind read some uncommitted data and like that you can run them at lower isolation levels. So, we get some if then they will actually run fast, they are not waiting for any locks to be really enter into lock. So, with that, let us close the session I come to the end of this slides such, but we can see that this I have given you an introduction to the various issues that are there.

In the concurrency control recovery which are of course, taking on as a taking at one module because there the issues are there into connected, but there are a lot more details in this particular topic and allows you to kind of spend time enjoying reading a little bit more about this fascinating thing called phantoms. So, with that we kind of come to the end of the course finally thank you.