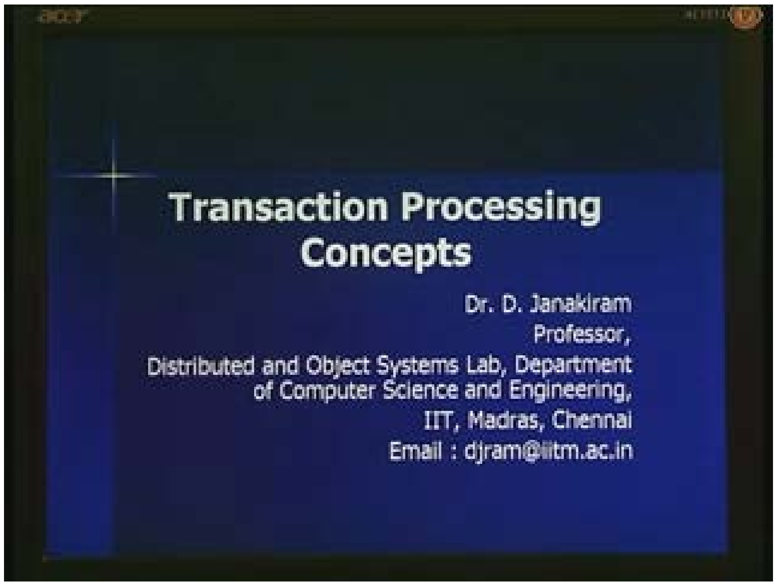
Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 17

Transaction Processing Concepts

In today's class we are going to look at transaction processing concepts. Any doubts in this particular class, you can e mail your doubts to the e mail address that is shown in the slide here djram@iitm.ac.in.

(Refer Slide Time: 1:40)

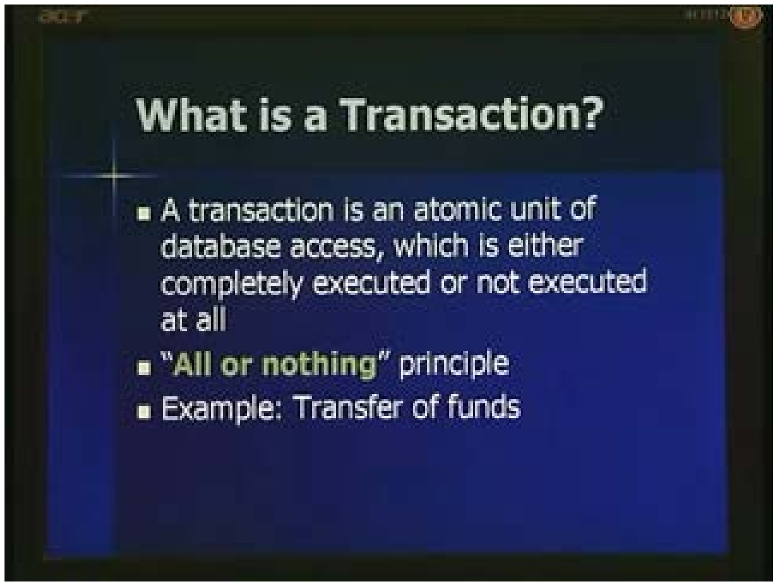


Transaction Processing Concepts

Dr. D. Janakiram
Professor,
Distributed and Object Systems Lab, Department of Computer Science and Engineering,
IIT, Madras, Chennai
Email : djram@iitm.ac.in

Now we will look at what is the meaning of a transaction in databases.

(Refer Slide Time: 1:52)



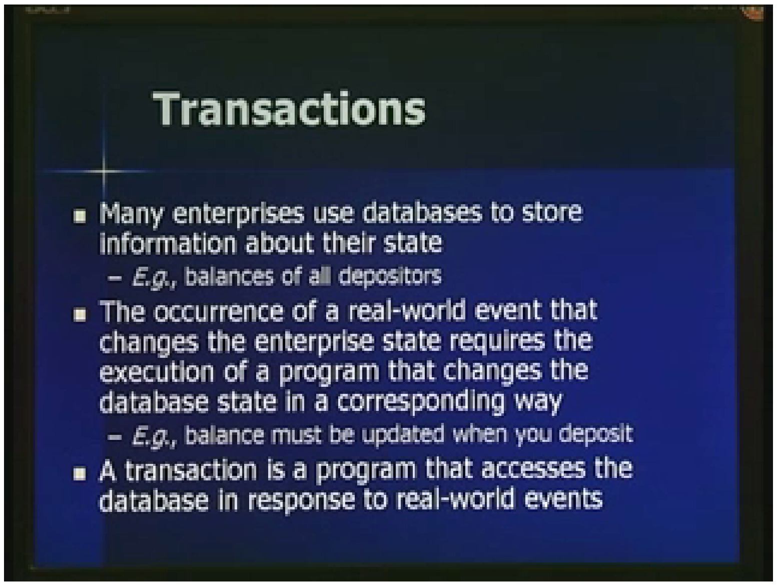
What is a Transaction?

- A transaction is an atomic unit of database access, which is either completely executed or not executed at all
- "**All or nothing**" principle
- Example: Transfer of funds

A transaction is essentially an atomic unit of access which is either completely executed or not executed at all. Typically databases store date of interest and we have applications trying to access the database and modify the data that is stored in the database. Now one example is banking database where you wish to transfer funds or withdraw amounts from your account in the bank database or you want to debit or credit certain amounts at transfer, amount from one account into another account. When the database is operated upon by the applications, we have certain instruction that will be executed and these instructions are not just normal instructions as executed in an operating system but they need to obey something more than the normal instructions.

What we are going to see is we are going to look at instructions and can see how these instructions need to be taken care of or what properties need to be enforced on these instructions which we call as transactions.

(Refer Slide Time: 03:26)

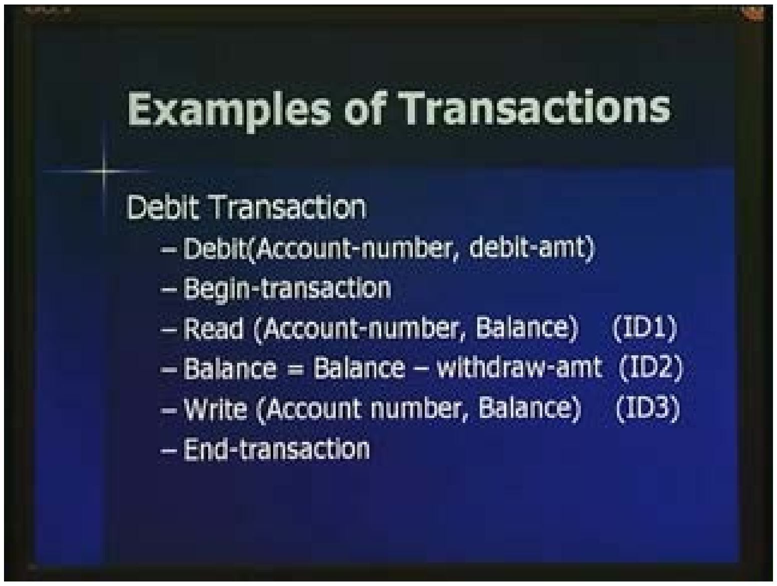


Transactions

- Many enterprises use databases to store information about their state
  - *E.g.*, balances of all depositors
- The occurrence of a real-world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way
  - *E.g.*, balance must be updated when you deposit
- A transaction is a program that accesses the database in response to real-world events

Now here are some examples of, typical examples of transaction. Many enterprises use databases to store information about their state. Now any occurrence of a real world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way. For example balance must be updated when you deposit into a banking database. When you withdraw an amount, basically need to update again your account by modifying the balance correctly. So typically what we say is the transaction is a program that accesses the database in response to real world events. They basically going to modify the state of the database. Transaction essentially modifies the state of the database.

(Refer Slide Time: 04:24)



Examples of Transactions

Debit Transaction

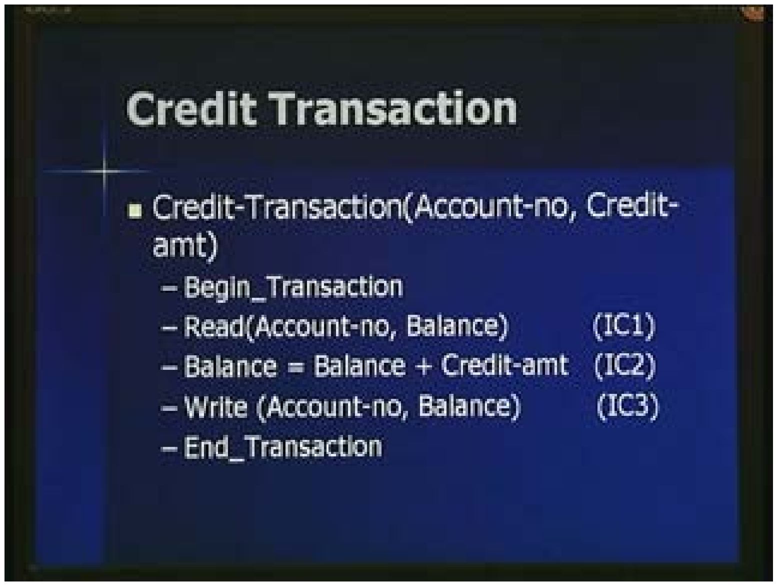
- Debit(Account-number, debit-amt)
- Begin-transaction
- Read (Account-number, Balance) (ID1)
- Balance = Balance – withdraw-amt (ID2)
- Write (Account number, Balance) (ID3)
- End-transaction

Now here is the very simple example of a database transaction. As you can see here it's a debit transaction which tries to withdraw a particular amount from a banking database. If we show that there is a debit, an account number and debit amount is given to the debit transaction. Now the begin transaction shows that this is the start of the transaction, execution of the transaction. Now there are three instructions which are part of this transaction. The first one is read the account number and the balance that is there in the database, in the banking database.

Now as part of the read instruction, we are going to see later, what are all the other instructions that need to be executed when this read instructions has to be executed by the transaction. Now after this read instruction succeeds, you will have your account balance in the variable called balance and now this balance has to be updated correspondingly with the amount that is going to be withdrawn from the account. As you can see later after the withdrawal amount, the new balance has to be computed by changing the balance and then the new balance has to be written on to the database back. And finally you signal that this transaction is finished by giving an end transaction. Now what we basically see here is the three instructions id 1, id 2 and id 3 shown in the slide here together constitute what we call as a transaction.

Now normal programs we don't basically distinguish by grouping instructions together and saying that they together constitute a transaction. Now we will go further deep analyze why these instruction put together will be called as transaction in this particular case.

(Refer Slide Time: 06:41)

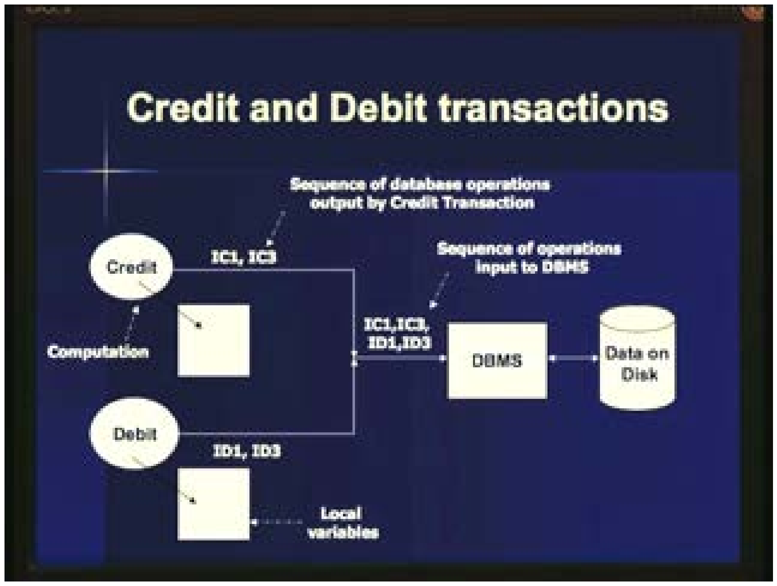


Credit Transaction

- Credit-Transaction(Account-no, Credit-amt)
  - Begin_Transaction
  - Read(Account-no, Balance) (IC1)
  - Balance = Balance + Credit-amt (IC2)
  - Write (Account-no, Balance) (IC3)
  - End_Transaction

Another example could be a credit transaction as you can see here again we have credit transaction being supplied with an account number and the amount that you would like to credit. As in the other case you have a begin transaction and end transaction signaling that all the instruction in between constitute together a transaction. Now as done in the earlier case the read instruction or read instruction is going to fetch the balance from your bank database, given the account number that particular account number, the balance will be read from the backend database. Now the new balance is computed by adding the amount that you crediting into the account and now you have to write back the new balance back into the database. Now these are very simple examples of what is the debit transaction and credit transaction in the case of a banking database.

(Refer Slide Time: 07:43)



Credit and Debit transactions

Sequence of database operations output by Credit Transaction
IC1, IC3
Sequence of operations input to DBMS
Credit
Computation
IC1,IC3, ID1,ID3
DBMS
Data on Disk
Debit
ID1, ID3
Local variables

Now we can see how these transactions in reality operate on the database. For example it is shown here in terms of the credit process which is shown in the slide shows the credit part of the transaction which we saw earlier. Now if you look at the debit part of the transaction you can see that the debit process executes the id 1, id 2, id 3 instructions shown earlier. As shown in this figure the backend database which is stored on the disc contains the information or the data relating to the bank customers.

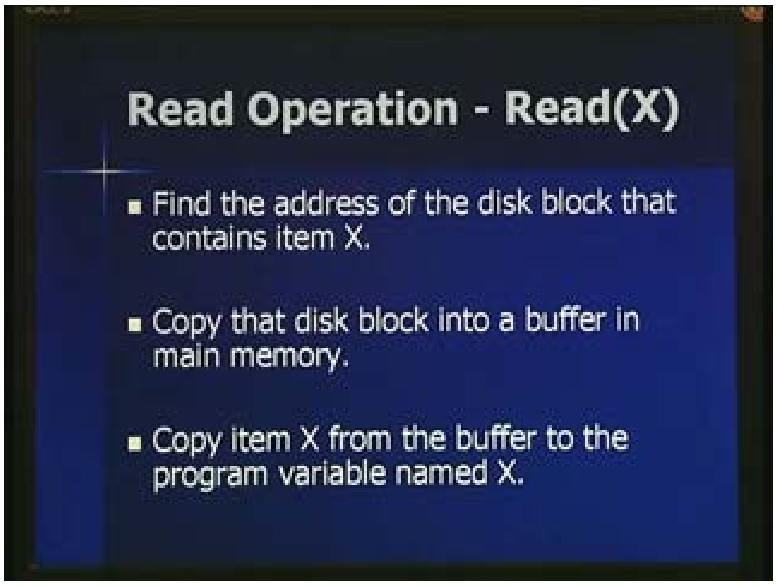
Now the credit or debit transaction needs to access the database and retrieve the information and correspondingly modify this information and after modification, they need to write the information back on to the database. Now let us understand this little more carefully here. Now we have has part of the debit transaction id 1, id 2 and id 3, id 1 is a read instruction. Now this read instruction has to go to the backend database management system which takes care of actually now finding out the corresponding data on the disc and move the data back to the local variables of the process, debit process.

That is now after the id 1 is executed, the balance variable will have correct data relating to the current balance that is there in the account number. Now since it is a debit transaction, the balance is going to be updated here by withdrawing the amount from the current balance and the new value will be computed which will be stored in the balance variable.

Now at the end of the transaction, debit transaction the value has to be written back onto the backend database and that what signals the end of the transaction. Now at the beginning of the transaction, the value is read from the bank database and at the end of the transaction, the new value is written back on the database this is what we mean by a transaction. A transaction essentially is reading some data from the database processing that data according to the semantics of the transaction and the new changed values are being written back onto the database at the end of the transaction. The same thing actually happens in the case of the credit transaction that is shown here except that the new balance is now added with the amount supplied.

Now we are going to see when the transactions are operating on the database, we need to have certain properties that needs to be enforced on these instructions so that the state of the database is in a consistent fashion. Now let us understand what are those properties and what happens really when the transactions are executing on the database.

(Refer Slide Time: 11:27)



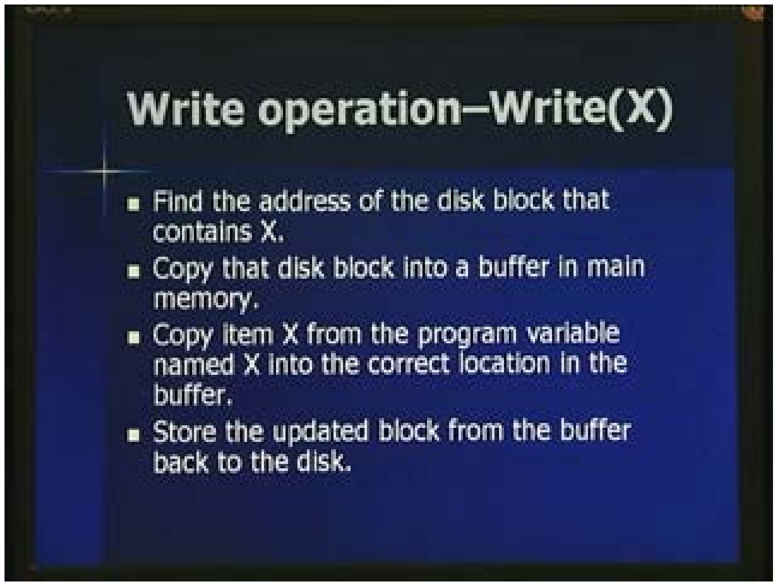
Read Operation - Read(X)

- Find the address of the disk block that contains item X.
- Copy that disk block into a buffer in main memory.
- Copy item X from the buffer to the program variable named X.

Now here is the case where we are explaining what happens when the transactions have to read data from the database. As shown earlier in the figure, we have to first find the address of the disk block that contains the data item x, x in this particular case can be a balance, it can be an account number. So it is basically the data that needs to be fetched from the backend database.

Now once you actually found the disk block, you have to copy the disk block information into a buffer in the main memory that is the local variable that is shown in the figure earlier. Now once the item has been copied into the buffer, the value of the disk block is copied into the buffer then you have to copy that value into the item x to show that the program variable named x now contains the value that is fetched from the disk block. This is what the read operation which is shown in the transaction signifies.

(Refer Slide Time: 12:43)



Write operation–Write(X)

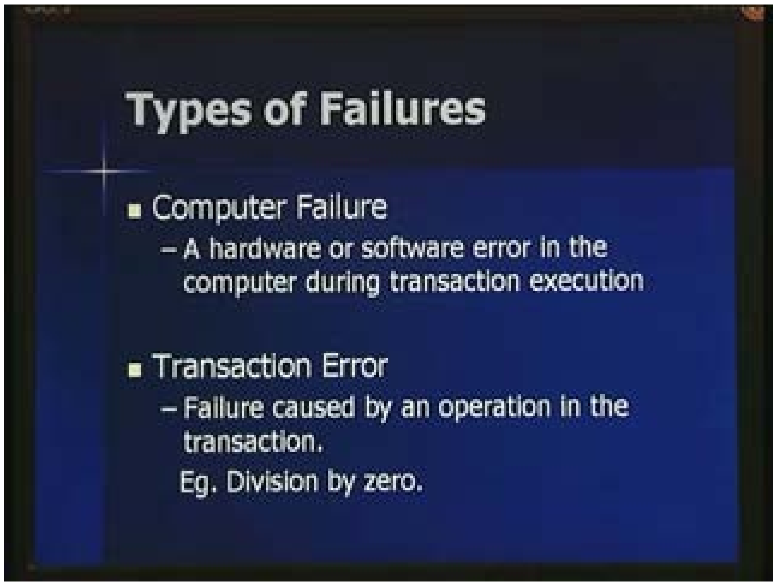
- Find the address of the disk block that contains X.
- Copy that disk block into a buffer in main memory.
- Copy item X from the program variable named X into the correct location in the buffer.
- Store the updated block from the buffer back to the disk.

Now we have correspondingly a write operation which shows how the write is executed. In this particular case you can see that find the address of the disk block that contains x. Now in this particular case you not only fetch the value of the data, you are going to modify the data and then the updated value is going to be written back onto the buffer as shown here. We can understand here, the last step is different from the earlier read operation. Store the updated block from the buffer back to the disk that is extra instruction that is going to be executed in this particular case.

Now essentially read and write are the two operations that are going to be used by the database transactions. And we have seen how the read and write fetch the required information for the database, fetch the required information for the transaction from the database, that's what was explained right now. This is similar to the normal process which would have read the value from a backend disk file. This is essentially same concept as of the process, trying to open a file and read the information from a file.

What more has to be done here is it is not just a set of file operation that are been performed by a process but we also need to enforce certain conditions on this instruction so that the database state is always maintained consistently. And that is what we mean by a transaction and the transaction has to enforce these properties on the instruction it is executing.

(Refer Slide Time: 14:48)



Types of Failures

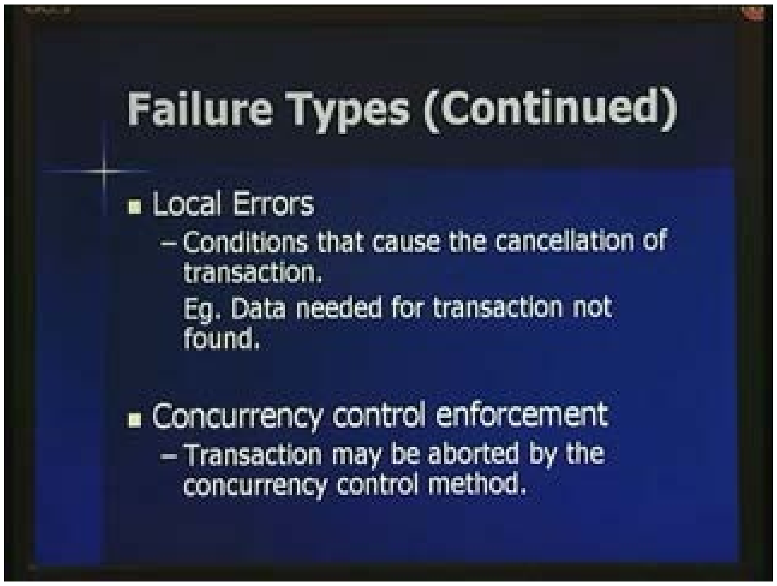
- Computer Failure
  - A hardware or software error in the computer during transaction execution
- Transaction Error
  - Failure caused by an operation in the transaction.
    Eg. Division by zero.

One of the things that will happen for a normal process is when failures occur, the process can leave the files that it has opened in a inconsistent state. For example if you understand the operating system and open, if your process opens files and the operating system crashes due to power failure or other reasons, what really happens is the files that are open could be left in an inconsistent fashion. Similarly if a file is been operated simultaneously by more than one user, again the chance of corruption exist on the file because there is no guarantees on how the file is been accessed simultaneously by multiple users. And this has to be prevented in a database because the data that is being stored in a database needs to be in a consistent fashion always that is to be maintained. For example if you consider a banking database, whatever happens you wish that you should not lose your money that you are depositing into the bank.

If the bank come and tells, there is a power failure and we have lost your 1 million rupees that you deposited, you are going to say that this quiet unacceptable and you want the bank to ensure that whatever happens, the data that is stored in the banking database is consistent all the time. This is an essential difference between database and file systems. File systems could be, they may not be any guarantees associated with file system in operating systems whereas when you take the database transaction there is certain level of guarantee that is given to you regarding the state of the database at the end of the execution of the transactions.

Now let us look at what kind of failures can happen in a system and what are the consequences of those failures. They could be different kinds of failures, we have actually listed a few failures here and we will start discussing them in more detail as we go along. Now you can see here the first kind of failure that can happen is a hardware or software error in the computer during transaction executing. This means it is possible that there is a problem in the hardware or the software. For example it could be a operating system bug or it could a hardware bug that could have made the computer fail which means that when you are executing this set of instruction that is shown earlier as a transaction, the failure can occur. The other kind of failure that can occur is internal to the transaction. Failure caused by an operation in the transaction, for example you are actually dividing by zero, divide by zero will cause the program to crash.

(Refer Slide Time: 18:23)



Failure Types (Continued)

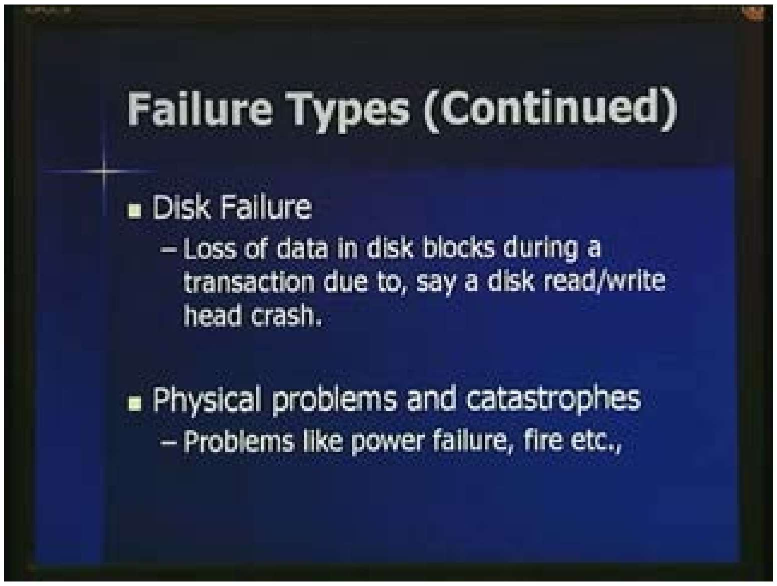
- Local Errors
  - Conditions that cause the cancellation of transaction.
    Eg. Data needed for transaction not found.
- Concurrency control enforcement
  - Transaction may be aborted by the concurrency control method.

So this is another kind of error that can happen in the system. The other kind of errors is condition that cause translation of a transaction. For example data needed for the transaction not found, you are trying to transfer funds from one account to another account then you recognize that the other account doesn't exist. This will result in the transaction to be aborted because the fund transfer is not happening correctly. The account into which you should be transferring the funds is not found in this particular case.

The other important issue is concurrency control enforcement which is to be done when multiple transactions are simultaneously operating on the database. For example if you really see how people can operate with the accounts, it's possible that two, you could be withdrawing some amount of money using your atm machine but at the same time there could be another transaction trying to transfer funds from your account to another account.

When this happens it is possible that the state of the database could be corrupted unless there is some kind of a concurrency control that is enforced to ensure that the system is in consistent state. We will go and look at the subsequent lectures in detail how the concurrency control is enforced by the database management system on transactions so that the database is in the consistent fashion. Now it is also possible because of the concurrency control that is enforced, a transaction is aborted because the transaction is started executing and the concurrency control mechanism found that transaction cannot proceed anymore. Then it may also abort the transaction. So this is another reason why a transaction could fail.

(Refer Slide Time: 20:37)



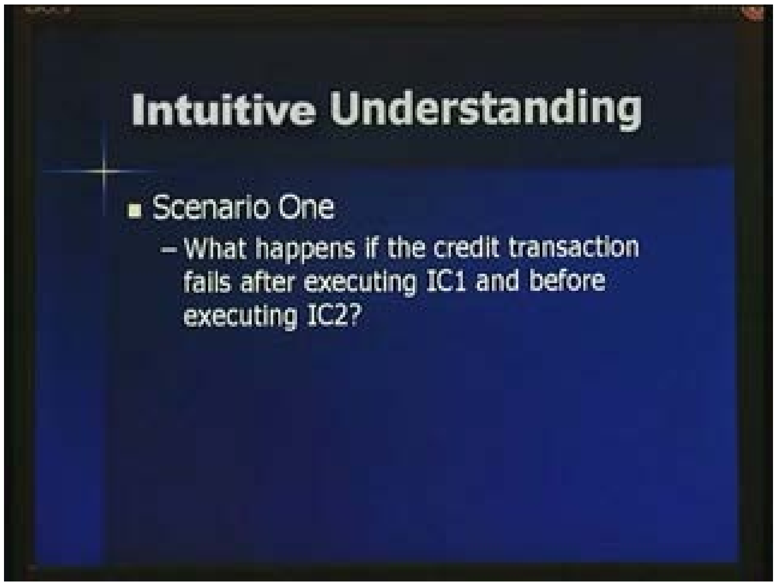
Failure Types (Continued)

- Disk Failure
  - Loss of data in disk blocks during a transaction due to, say a disk read/write head crash.
- Physical problems and catastrophes
  - Problems like power failure, fire etc.,

They could be other reasons as well, something like loss of data in the disk blocks during a transaction, due to let us say the disk head has crashed. So then it is possible that you are not able to retrieve the data correctly from the disk. This is a disk failure. This occurs when disk hard disk has failed. There could be other catastrophic reasons for you when you deal with databases, things like power failure, fire and other kinds of catastrophes like earthquake which could destroy the data and they are beyond the human control.

And one of the things you must realize is all these kinds of failures are possible and in the event of this failures, the database till should ensure that the data that is stored in the database is consistent and it is available by other means. That is you are able to retrieve the data back, even when failures of this nature occur. Now a part of this lecture we will explore, how you can handle this situation when failures occur when transactions are executing.

(Refer Slide Time: 22:06)



Intuitive Understanding

- Scenario One
  - What happens if the credit transaction fails after executing IC1 and before executing IC2?

Now what we are going to look at is we will intuitively understand the concept of transaction to start with and we will see what can happen to a transaction in the event of failures. We in fact look at several kinds of failures, starting from a transactional error to a disk failure, to a power failure, to a more catastrophic failure. So what we would like to see is what happens if this failures occur, when transactions are in progress. Imagine you are withdrawing money from your bank account and the power fails.

Now what happens? Is your bank account still shows correct balance or is it going to show that you have already withdrawn when you have not taken your amount. What is the state in which your bank database will be left when the failures occurs, when your withdrawing money from a account. Now here is a case where it is shown more precisely to say what kind of scenarios can prevail and how those scenarios have to be addressed.

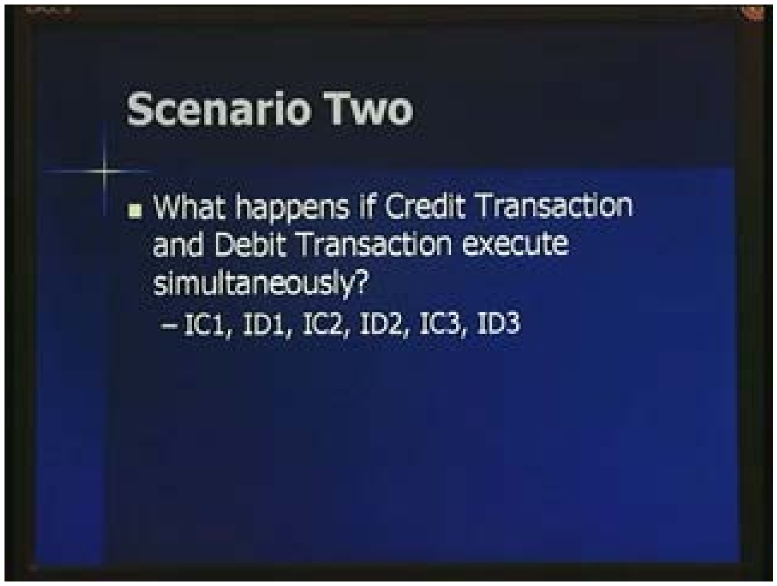
Look at the scenario one. What happens if the credit transaction fails after executing ic 1 and before executing ic 2? Remember ic 1 is a read account balance instruction and ic 2 is when it is actually modifying that balance local which still has not written that value back onto the database because the right has to be done in the end of it, ic 3 has to be done to write the balance back on to the backend database.

Now what happens if the credit transaction fails after executing ic 1 but before executing ic 2? Now in a normal scenario, if you don’t really take care of this situation, it is possible that the database is left in an unknown and undeterministic condition when the failure occurs. But you have to actually prevent this from happening, by saying that it will bring back the database to a consist fashion if the failure occurs. In this particular case, you have to ensure that all the instructions ic 1, ic 2 and ic 3 are either executed or not executed at all.

This is a very important property that needs to be ensured for database transactions. We will go to see this property in more detail, this property is called the atomicity property of the transactions. That is all the instruction put together have to be executed either in full or none of them should executed at all. In fact if you carefully look at the initial example where we had proceeded the three instructions ic, 1 ic 2 and ic 3 with begin transaction and end transaction. All the instructions between begin and end have to obey this property called the atomicity property that either all the instructions are executed in full or none of them are executed.

This is what we see has scenario one. What can happen if the credit transaction fails after executing ic 1 and before executing ic 2. Let us move to scenario two. What happens in scenario two?

(Refer Slide Time: 26:01)



Scenario Two

- What happens if Credit Transaction and Debit Transaction execute simultaneously?
  - IC1, ID1, IC2, ID2, IC3, ID3

If the credit and debit transaction executes simultaneously, what are the likely things that can happen? In fact shown a case where ic 1 is executed then followed by id 1 is executed then ic 2 is executed then id 2 is executed, then ic 3 and followed by id 3. If you carefully look at the way it was written here, both the credit and the debit instructions have been interleaved. ic 1 is basically a read account number and the balance, id 1 is also a read instruction on the database except that this is the debit instruction, the earlier is the credit instruction. Now this will also read the balance in the account number.

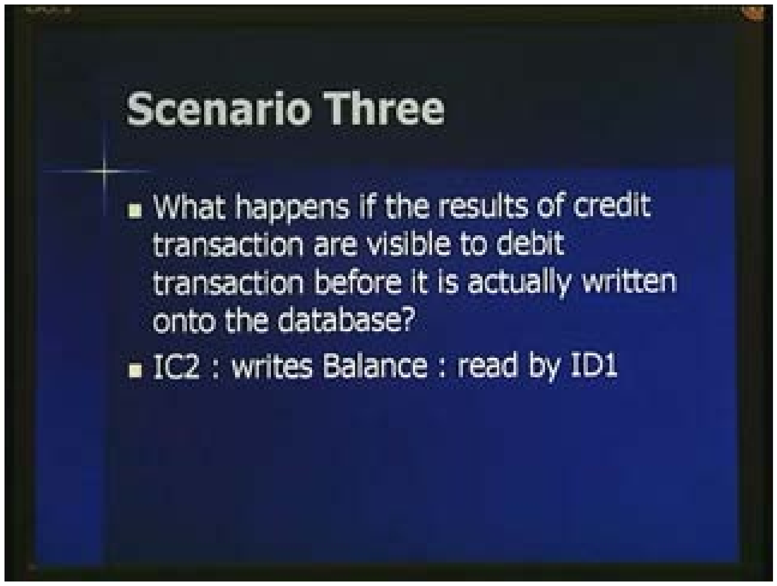
Now if you say that both are operating at the same account number, they are reading the balance at the same time. now imagine ic 1 has the same balance, the value that is currently let us say the account has a balance of 500 rupees in your account then both ic 1 and id 1 read the value as 500. Now imagine that your depositing 200 rupees and withdrawing 100 rupees. Now ic 2 will say that 500 plus 200 which is actually 700 and ic 3 will try writing 700 back into the database whereas id 2 will try to reduce the balance from 500 by 400 and id 3 will write the value has 400.

Now you can see, you have lost some amount in the process because the credit amount is completely lost because both the credit and the debit transactions are simultaneously operating and only the debit is shown here, the credit is lost. The credit that is done into the database is lost in this particular case. So this is what we see as scenario two. When transaction operate concurrently on the database items, it is possible that the database state is left in an inconsistent fashion as shown in this particular example. Now we have to prevent this from happening and this is what we call as the consistency property of the transaction.

Now what we mean by consistency here is when the transactions are operating concurrently, simultaneously we need to enforce the condition that the transaction in effect have executed one after the other rather than simultaneously. This is in some sense we need to prevent if there is conflict between transactions. They operating simultaneously on the database items has to be prevented and this is achieved by what we earlier called as concurrency control mechanisms

So we need concurrency control mechanisms for making sure that the database, when it is operated upon simultaneously by multiple transactions is not left in an inconsistent state. this is what we see has a scenario two and scenario two gives the property of consistency whereas scenario one gives the property of atomicity to the transaction.

(Refer Slide Time: 30:00)



Scenario Three

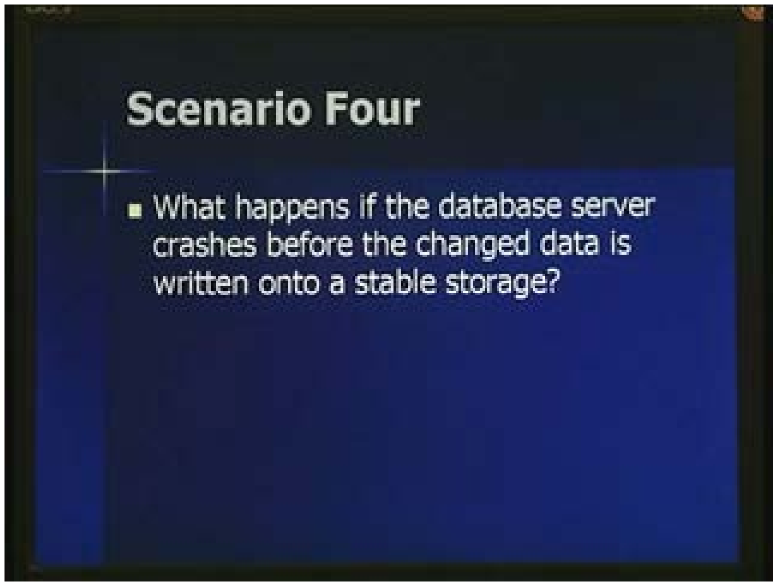
- What happens if the results of credit transaction are visible to debit transaction before it is actually written onto the database?
- IC2 : writes Balance : read by ID1

Now let us move on to the third scenario. Scenario three where basically it is possible for one transaction to see the values of the other transaction before it is actually finished, its full operation. Now that what is actually stated here. What happens if the result of the credit transaction are visible to debit transaction, before it is actually written onto the database? What does this mean? This is elaborated further by saying that ic 2 writes the balance. The earlier case if you are actually depositing 200 rupees when your initial balance is 500, ic 2 will write the value of 700.

Now the debit can read this value of 700, even before the credit has actually committed its value to the backend database. Now we can, debit can now go and then withdraw the money from the new balance even before it is written back on to the backend database. If this happens, the results of one transaction are visible. In this particular case, the credit transaction are visible to the debit transaction before it is actually finished execution. Now this results in what we call as an isolation property because for some reason if the credit transaction fails later, for various other reasons if the credit transaction fails and if its results are already visible for debit transaction, you need to abort the debit transaction also because it has read the values of a transaction that is aborted. This is what we mean by causing cascading aborts.

If a transaction values or results are available for some other transaction before it is committed, it could lead to cascading aborts. To avoid this, what we have to do is we have to enforce the property called isolation. Isolation ensures that the transaction results, the values which the transaction has changed, the values of the data items which a transaction has changed are not available for other transactions till the transaction has actually committed. Actually concurrency control protocols and commit protocols which go together, ensure the consistency of the database in the presence of multiple transactions executing simultaneously on the database. Now let us move on to the forth scenario which leads to the final property, forth property of the transaction.

(Refer Slide Time: 33:17)



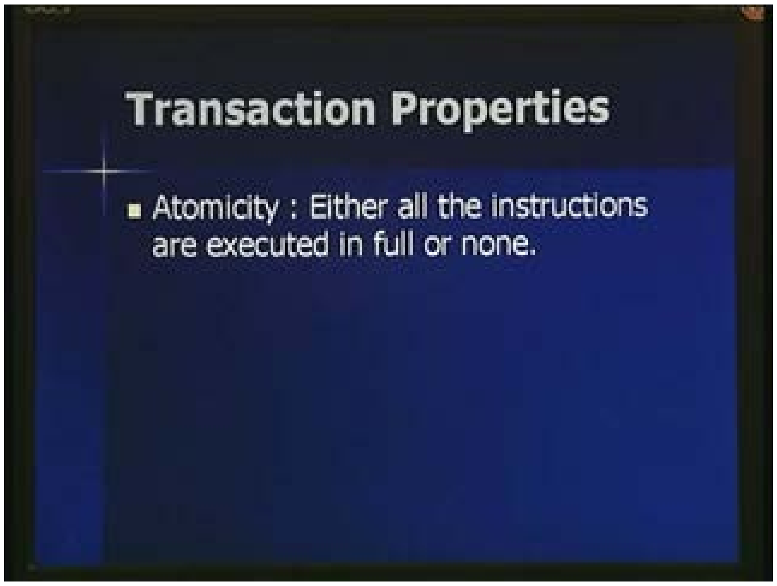
Scenario Four

- What happens if the database server crashes before the changed data is written onto a stable storage?

Now this scenario four tells what happens if the database server crashes before the changed data is written onto a stable storage. One could imagine several situations where the database values have been written, the transaction have committed but their final values have not been written onto the database for various reasons. Now whatever happens after the transactions says it has committed, its value should be preserved. The value that the transaction has changed should never be altered after the transaction has been committed.

So you have no way of saying that the results of the transaction is lost after it has committed. This is what we mean by the property of durability. All results of the committed transaction are preserved after that point once the transaction has committed. You have to guarantee this inspite of any other kind of failure that may happen to a database.

(Refer Slide Time: 34:39)

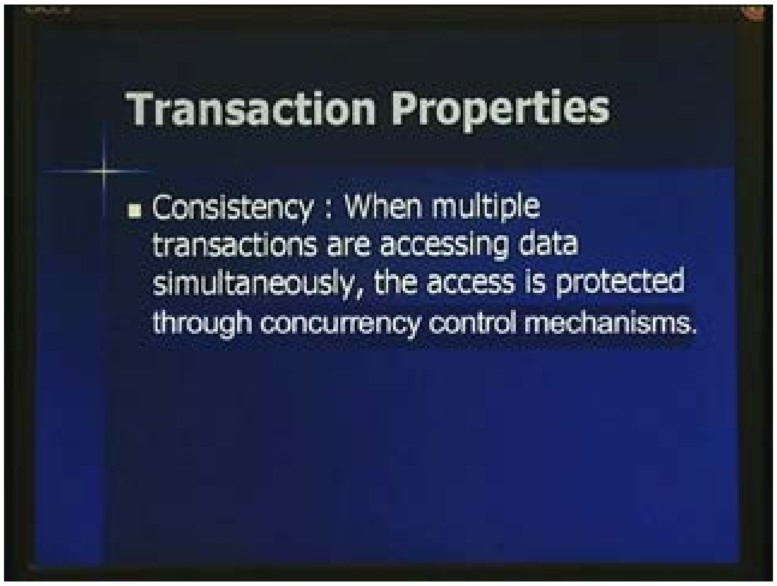


Transaction Properties

- Atomicity : Either all the instructions are executed in full or none.

These four properties are very important properties when we talk about transactions. Now to just repeat this properties, atomicty ensures which we actually derived from scenario one ensures that all the instructions of a transaction are executed in full or none. So the first question of some part of the instruction being executed and some part of the transaction instruction not being executed doesn't arise at all. Because we ensure that all the instruction of the transaction are executed in full or none.

(Refer Slide Time: 35:23)

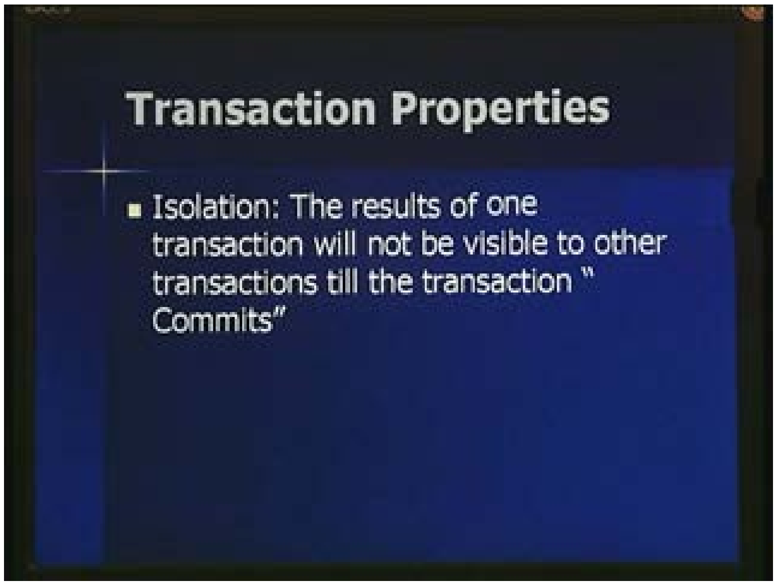


Transaction Properties

- Consistency : When multiple transactions are accessing data simultaneously, the access is protected through concurrency control mechanisms.

The second property which we discussed is a consistency property. When multiple transactions are accessing data simultaneously, the access is protected through concurrency control mechanisms to ensure that the updates which are done by the concurrently executing transactions are not lost on the database. This is what we actually mean by the property of consistency. And we also mention that the consistency is ensured in database management systems by using a set of concurrency control protocols and we are going to study this concurrency control protocols in depth during this lectures.

(Refer Slide Time: 36:10)

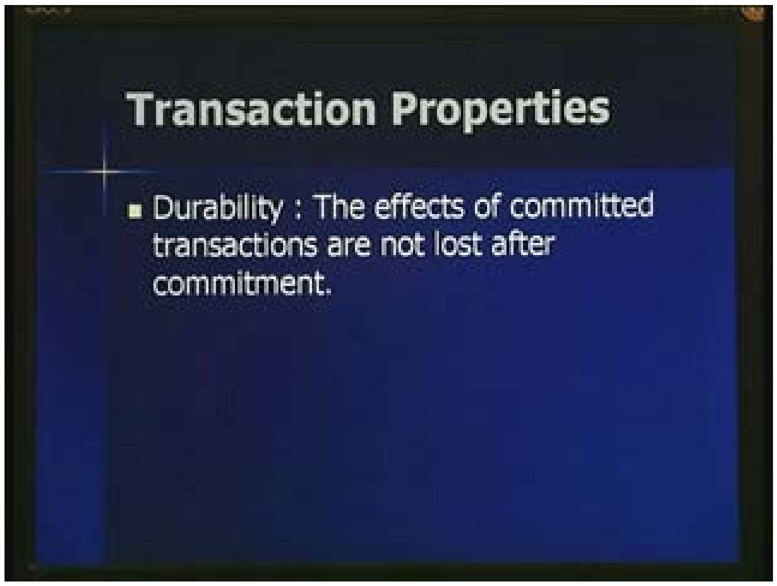


Transaction Properties

- Isolation: The results of one transaction will not be visible to other transactions till the transaction " Commits"

Now the third property is isolation. The isolation property ensures that the results of one transaction will not be visible to the other transaction till the transaction commits. This ensures that there are no problems relating to partial results being available for other transactions. We also mention that when this happens cascading aborts takes place, when one transaction results have been read by other transaction and the earlier transaction has to be aborted. And to prevent this cascading aborts, we enforce the property of isolation on the transactions.

(Refer Slide Time: 36:58)

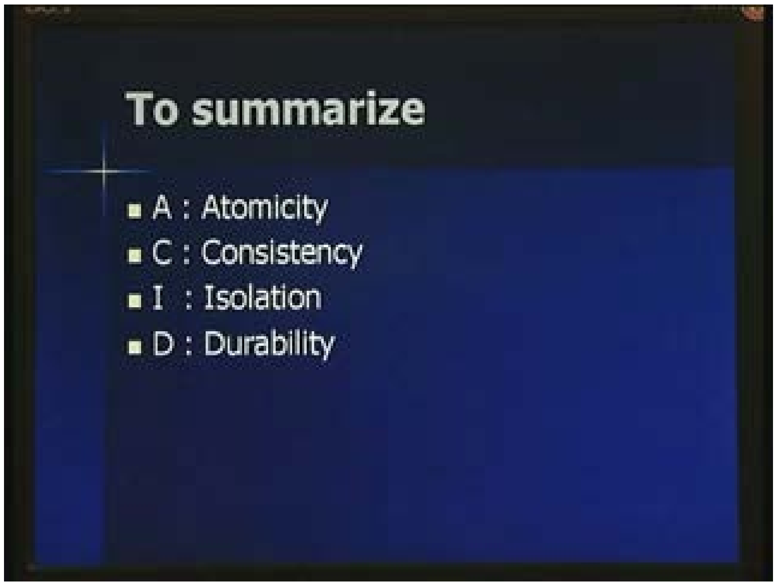


Transaction Properties

- Durability : The effects of committed transactions are not lost after commitment.

The forth property is the durability property and it says that the effects of the committed transactions are not lost after commitment. For example if you have deposited some amount into your bank and you want to ensure that it’s never lost after you have actually deposited the money into the bank. It will never be lost, that is basically the durability property.

(Refer Slide Time: 37:26)

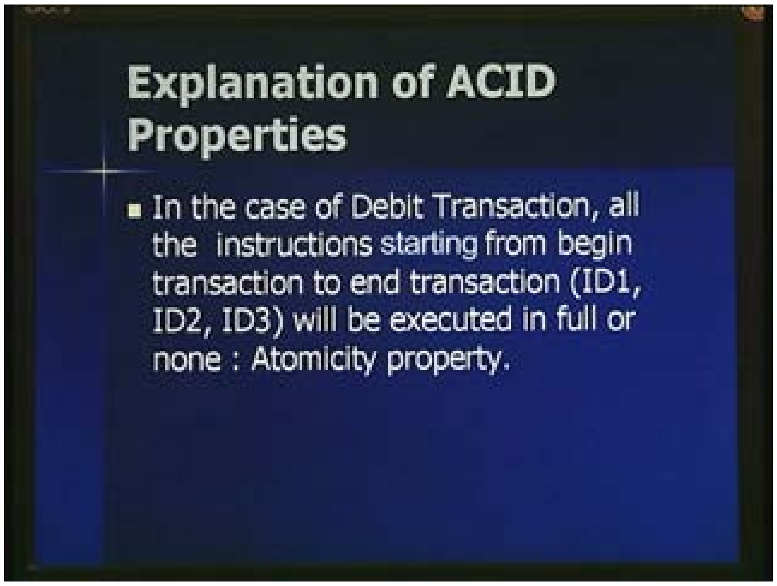


To summarize

- A : Atomicity
- C : Consistency
- I : Isolation
- D : Durability

Now all these four properties put together are nicely known as the acid properties of the transaction as show here. It is the summarization of the four properties that we have been so far discussing. A stands for atomicity, C stands for consistency, I stands for isolation and D stands for durability. So these four properties put together are called as the acid properties of the transaction. And normal process will not obey this acid properties whereas the transactions in the database context will obey this acid properties. Now one of the things that we are going to look at through this lectures is see how this acid properties are realized by the database management system when we are actually executing transactions in the database.

(Refer Slide Time: 38:27)



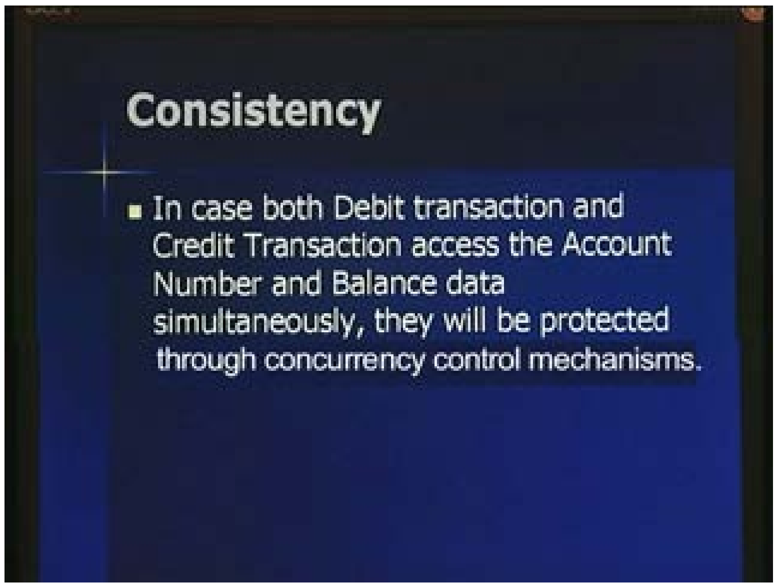
Explanation of ACID Properties

- In the case of Debit Transaction, all the instructions starting from begin transaction to end transaction (ID1, ID2, ID3) will be executed in full or none : Atomicity property.

Now we will also further elaborate this acid properties littler more formally by actually taking what happens and how this acid properties are ensured. Now as you can see here in the case of the debit transaction, all the instructions starting from the begin transaction to the end transaction will be executed in full or none which actually means that id 1, id 2, id 3 have to be executed in full. Now one of the things we are going to do is when there is a begin transaction, we record the state of the database. Now whatever happens after the transaction starts executing, if there is a failure we will ensure that you get back to that state by restoring the state to the original state if the instructions are not executed in full.

For example if your original balance, starting balance is 500 rupees and for some reason the debit transaction cannot be executed, all the instruction restore the balance back to 500. This is what we mean by undoing a transaction. The transaction, all the instructions which were executed partially, till completion of the transaction are rolled back which means all those instruction will be nullified. We actually rollback on those transactions, so that effect on those instruction is nullified. So this is what we mean by atomicity property. We ensure that either all the instructions are either executed together or none of them are executed.

(Refer Slide Time: 40:17)



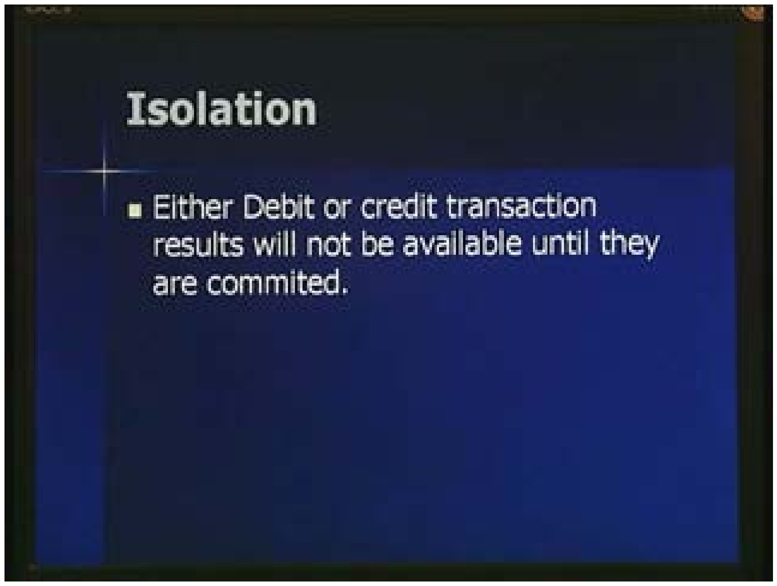
Consistency

- In case both Debit transaction and Credit Transaction access the Account Number and Balance data simultaneously, they will be protected through concurrency control mechanisms.

Now again, to stress again what really we were talking about consistency in case of both debit transaction and credit transaction access the balance data simultaneously, we will protect them through the concurrency control mechanisms. A simple mechanism that we are going to use is we lock the database items and allow only transaction which acquires this locks to change the values of those data items. And only when the transaction releases the locks on those data items, other transaction will be allowed to use those data items, this is a very simple technique.

The most sophisticated techniques that can be used for enforcing concurrency control mechanism but this is what we would like to do to ensure that the consistency property is enforced or realized on the database.

(Refer Slide Time: 41:14)

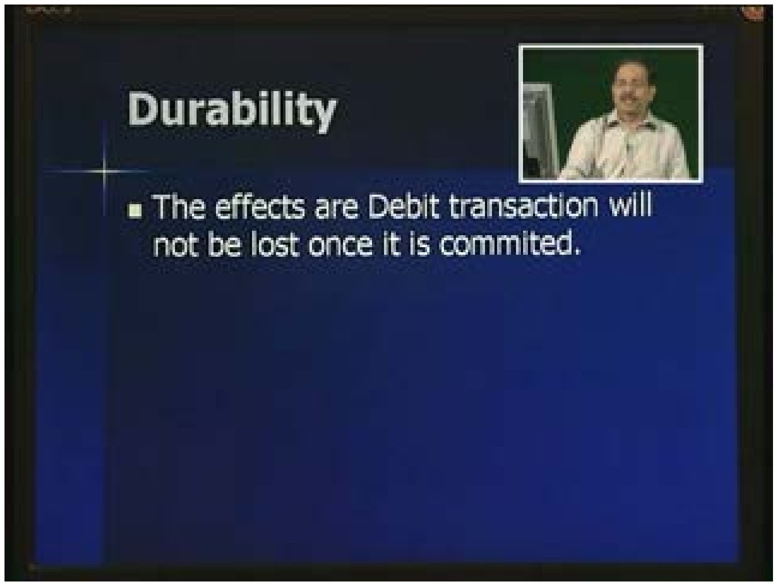


Isolation

- Either Debit or credit transaction results will not be available until they are commited.

Now when you go to the isolation property, you are going to look at either debit or credit transaction results will not be available unless they are committed. in one way these transaction have to hold on to this locks when they require and should not release those locks for other transactions till they are committed to ensure the values that they are modify are not available to other transactions till they finished execution, till they reach the state of end transaction which means that now they have committed their values and after that only those results will be visible for other transactions.

(Refer Slide Time: 41:56)

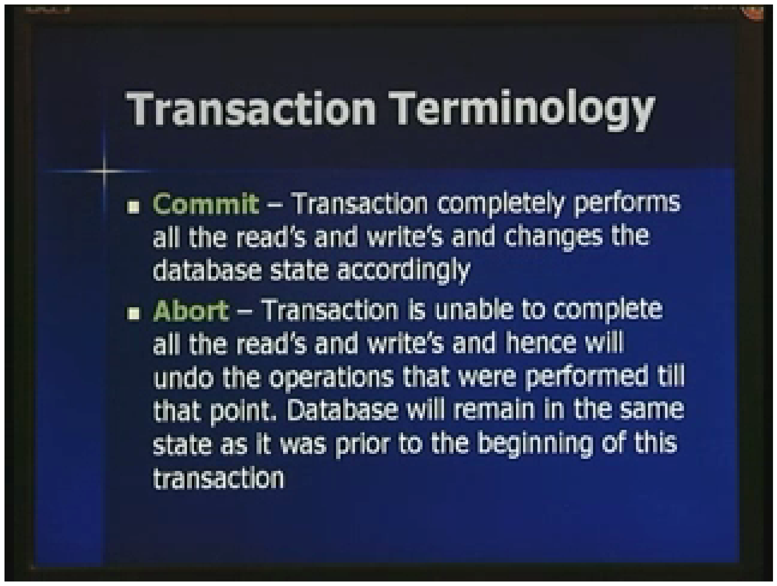


Durability

- The effects are Debit transaction will not be lost once it is commited.

Now there are several ways in which we can ensure the durability property. the durability property will ensure that you have backups sufficient backups, you have written all your logs committed transaction locks and there are various ways in which the effects of this are preserved to make sure that all the committed transaction values can always be obtained by using the backups and the transaction logs. You are going to look at this property and how this is realized in detail as we go along.

(Refer Slide Time: 42:31)



Transaction Terminology

- **Commit** – Transaction completely performs all the read's and write's and changes the database state accordingly
- **Abort** – Transaction is unable to complete all the read's and write's and hence will undo the operations that were performed till that point. Database will remain in the same state as it was prior to the beginning of this transaction

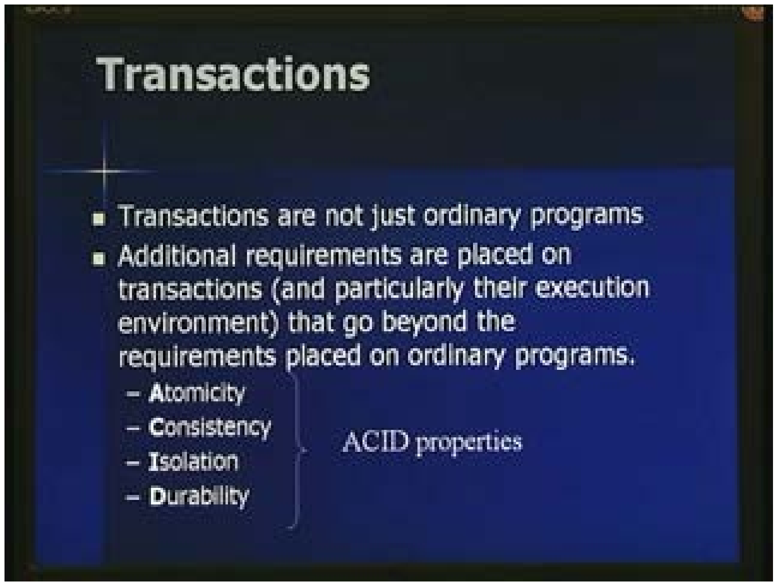
Now we come into more details of what really happens with transactions. For this actually we will introduce certain terminology to start with.

The idea is to get more formal with the transaction concepts, see them in more detail as we progress. So far we have been looking at the properties very intuitively, trying to understand them in a very intuitive fashion. Now we will try to understand the concepts in the more formal way.

Now there are two case in which the transactions can enter into. one is a commit state which actually means when the transaction has completely executed all its instructions, it can enter into a commit state which actually means that all the reads and writes of the data items which is actually read can now be written back and there been safely written back of the database in which case we say that the transaction has committed itself. Now for some reason, the transaction has started executing but it cannot commit the values of the data item that it has changed which it has read from the database then we say that the database has entered the state of abort which actually means that all the effects of the transaction will be nullified and database state will be left when the transaction start executed.

That is equivalent to saying that I actually had a begin transaction. The state of the database when I started executing this transaction which is the begin transaction and I actually keep the state back to that initial state when the transaction started executing. That is called the abort state. So we have commit and abort. A transaction could be either committing or aborting. When it is says it has committed, it is writing all the values that it has read and changed back onto the database. When it is says it is aborting, it is not committing any of the values that it has changed.

(Refer Slide Time: 44:46)



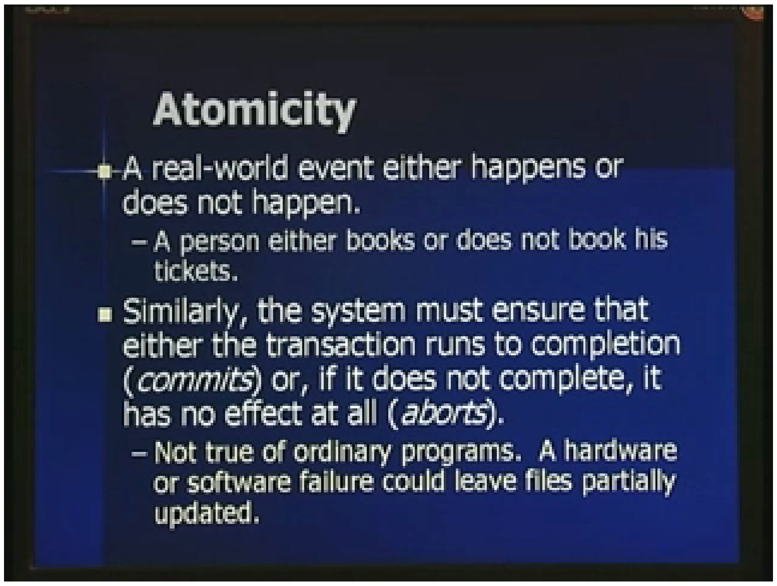
Transactions

- Transactions are not just ordinary programs
- Additional requirements are placed on transactions (and particularly their execution environment) that go beyond the requirements placed on ordinary programs.
  - Atomicity
  - Consistency
  - Isolation
  - Durability

ACID properties

So to emphasis the transactions are not just ordinary programs instructions, all our discussion today highlights the fact that additional requirements are placed on transactions to ensure that this acid properties atomicity, consistency and isolation are realized with the transactions.

(Refer Slide Time: 45:14)



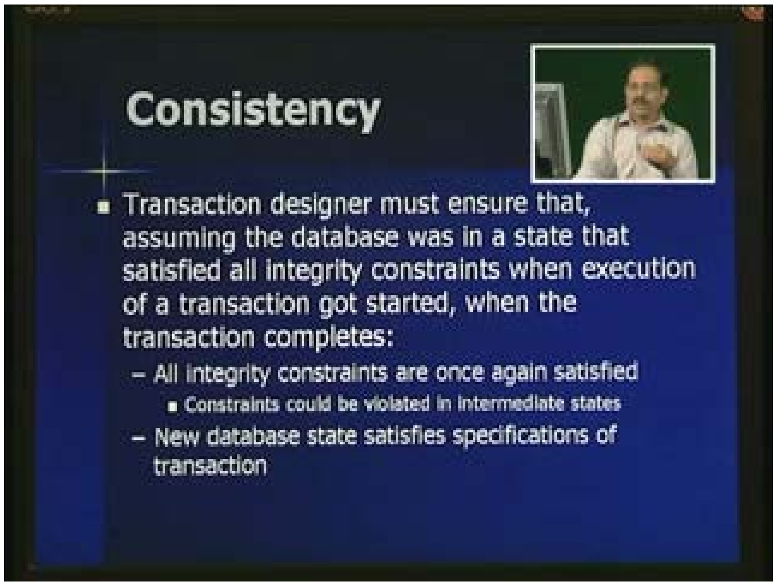
Atomicity

- A real-world event either happens or does not happen.
  - A person either books or does not book his tickets.
- Similarly, the system must ensure that either the transaction runs to completion (*commits*) or, if it does not complete, it has no effect at all (*aborts*).
  - Not true of ordinary programs. A hardware or software failure could leave files partially updated.

Now here is the case where we can quickly see with respect to commit and abort, what really happens for the atomicity property. Now atomicity property says that a real world event either happens or does not happen. Now if you take the case where a person either books or does not book his tickets which actually means that when he is actually book his tickets, the transaction has committed the values. When he says he didn’t book the ticket, it means that the transaction has aborted. And again to give the state the think that it is not true of ordinary programs, a hardware or a software failure could leave files partially updated which is not the case in the case of transactions.

When you say I have booked my ticket, it means that you book your ticket that means the transaction booking tickets as committed and when it says it is not book the ticket means that it is not committed, it is aborted.

(Refer Slide Time: 46:26)

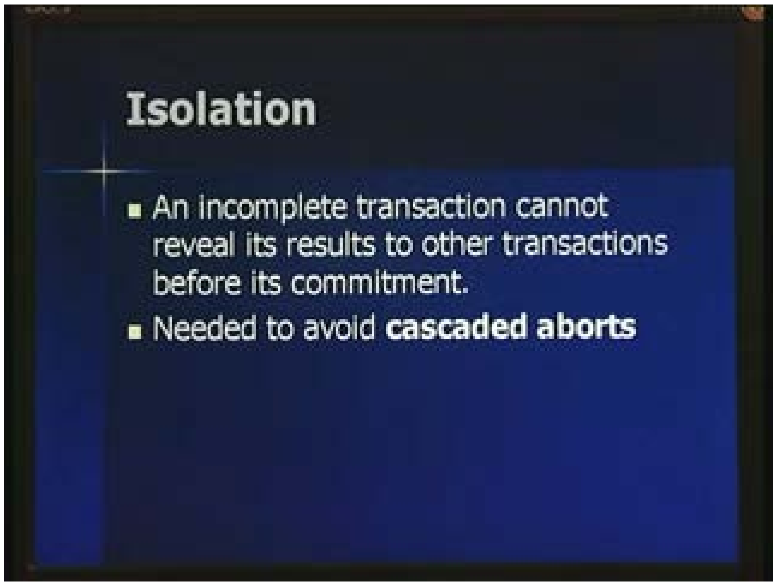


Consistency

- Transaction designer must ensure that, assuming the database was in a state that satisfied all integrity constraints when execution of a transaction got started, when the transaction completes:
  - All integrity constraints are once again satisfied
    - Constraints could be violated in intermediate states
  - New database state satisfies specifications of transaction

Now coming to the consistency property, you have to ensure that the set of integrity constraints that are specified by the transaction are all enforced when the transaction has executed. Now what we mean by this is transaction designer must ensure that assuming the database was in a state that is satisfied all integrity constraints when execution of a transaction got started. Then when the transaction has actually completed execution, we need to ensure that all the integrity constraints are once again satisfied. In a simple way, the consistency can be ensured by saying that the transaction execution results in a serializable execution. That is the transaction is executed as if all the operation have been executed in a serial fashion.

(Refer Slide Time: 47:19)

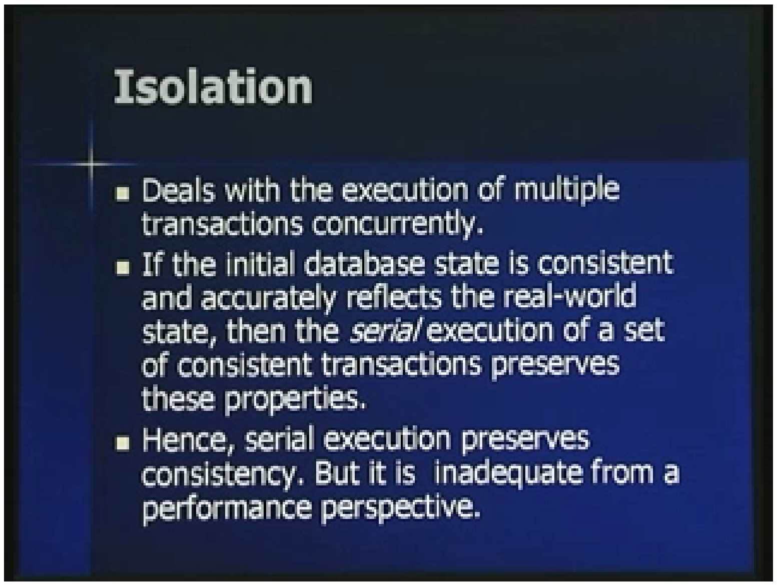


Isolation

- An incomplete transaction cannot reveal its results to other transactions before its commitment.
- Needed to avoid **cascaded aborts**

We are going to look at that particular property right now in the next few minutes in the little more detail and see what does it mean by consistency preservation. And isolation essentially will avoid cascaded abort as explained earlier.

(Refer Slide Time: 47:36)

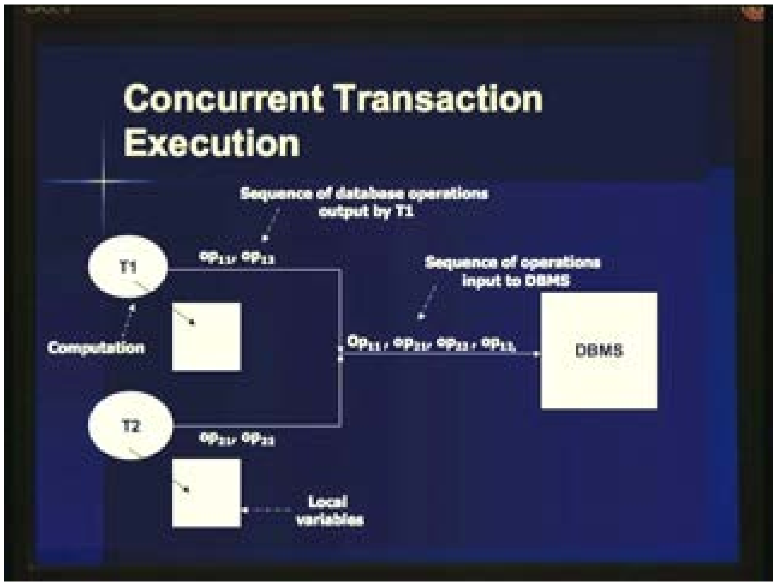


Isolation

- Deals with the execution of multiple transactions concurrently.
- If the initial database state is consistent and accurately reflects the real-world state, then the *serial* execution of a set of consistent transactions preserves these properties.
- Hence, serial execution preserves consistency. But it is inadequate from a performance perspective.

Here is a simple case where isolation was given in a more detailed fashion. It relates to when multiple transaction execute concurrently and you want to actually ensure that the final execution thus preserves the consistency by ensuring that one transaction values are not read by the other transaction till it has finished.

(Refer Slide Time: 48:09)

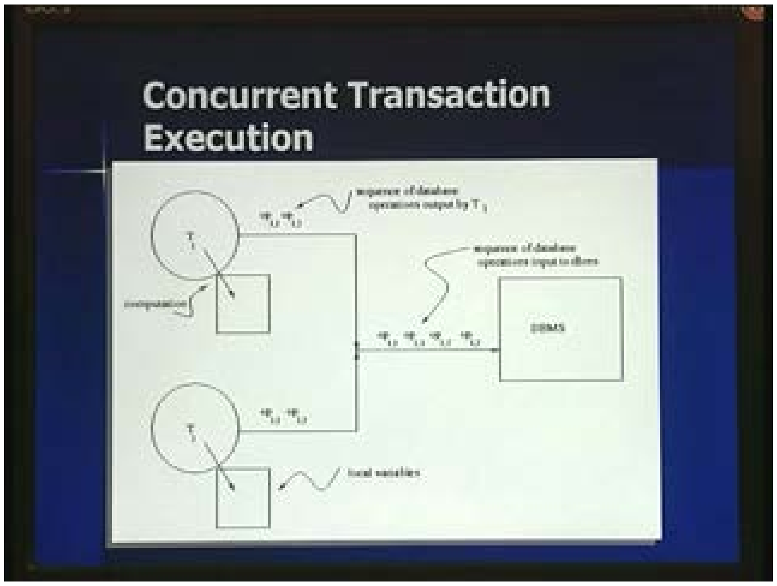

Concurrent Transaction Execution
Sequence of database operations output by T1
T1
op_{11}, op_{12}
Sequence of operations input to DBMS
Computation
$op_{11}, op_{21}, op_{22}, op_{12}$
DBMS
T2
op_{21}, op_{22}
Local variables

Now we will see this property of concurrency and isolation together by taking a simple example as given in this particular case. As explained in this diagram, as you can see this here t_1 has two operations op 1 and op 1 2 and t_2 has two operations operation 2 1 and 2 2. Now it's possible that these sequence of operations can be interleaved in multiple ways on your database. As you can see here one possible sequence is operation 1 1 executed first of transaction 1 and then operation 2 1 of transaction of transaction 2 then operation 2 2 is executed of transaction 2 and operation 1 2 of first transaction is executed. As you can see here from the execution sequence of this, we may not be able to say that transaction t_1 completed all its operations before transaction 2 is executed but on the other hand if there is a way, you can ensure that all the instructions of transaction 1 are finished before transaction 2. It is equivalent to saying that the set of instructions that are executed are all in a serial fashion.

Now one of the requirements of operation 1 1 and operation 1 2 to be serializable in this context is they are not operating on the same account. It is possible that two people are withdrawing money from two different accounts. Since there is no conflict in this particular case between the two operations, it doesn't really matter even if $o_{1\ 2}$ is executed later. we can always say interchanged the operations since there is no conflict in this particular case and rewrite the set of operations as if they have been executed as $o_{1\ 2}$, $o_{1\ 1}$, $o_{1\ 2}$ and then $o_{2\ 1}$ and $o_{2\ 2}$. When operation don't conflict, it is possible for us to interchange operations and then ensure what we see the property of serializability. That is all the operations of t_1 have finished before t_2, that is what we mean by serializability or in other sense what we going to say is all the operations of t_1 since they are finished, t_1 precedes t_2 in terms of the operations have been executed on the database. This is called serializability.

However let us say that the operation 1 1 and operation 2 1 conflict with each other in the sense that they access the same database item. In this particular case, we can say that they are accessing the same account and the same balance and they are trying to modify the same balance. They are not just reading but writing values, modified values onto the database in which case we say there is a conflict. Now operations will conflict if they operate on a same data item and one of them is right that is what we mean by conflicting operations. When transaction is conflict we need to serialize the transactions and this is what we mean by conflict serializability. the conflicting operations should be executed in such a way that we know that the conflicting operations are executed in a serial fashion which actually means that we can ensure that the operations on the database in this particular case if $o_{1\ 1}$ and $o_{2\ 1}$ have conflicting there have been executed one after the other and that decides how the transaction precede with each other.

(Refer Slide Time: 52:33)


Concurrent Transaction Execution

sequence of database operations output by T1
sequence of database operations input to dbms
computation
DBMS
local variables

Now let us see the following scenario where on $o_{1\ 1}$ and $o_{2\ 1}$ which are two conflicting operations, we say that t_1 preceded t_2 and let us say $o_{1\ 2}$ and $o_{2\ 2}$ are also conflicting. Now let us say as far as those operations are concerned, t_2 precedes t_1. Now this is a scenario that will result in the transactions t_1 and t_2 not being serializable because as far as the conflicting operations $o_{1\ 1}$ and $o_{2\ 1}$ are concerned, the t_1 is preceding t_2. And in the case of $o_{1\ 2}$ and $o_{2\ 2}$ which are again conflicting the transactions are preceding in the other direction, t_2 is preceding t_1. So we can't say as far as the conflicting operations are concerned t_1 is executed before t_2, one case t_1 executed before t_2 in the other case t_2 is executed before t_1. This is a very interesting thing which we going to study in detail in the next lecture.

We see how transactions need to preserve the property of conflict serializability. Only when transactions execute and they are serializable, conflicts serializable we say that the database is, the transactions have executed in the correct fashion on the database. We are going to further study this property in tomorrow's lecture in detail as concurrency control mechanisms. The concurrency control mechanisms are expected to provide the property of conflict serializability. They ensure that when transactions are executing concurrently, we can serialize them in a, transactions are serializable and that's the property that is ensured by concurrency control mechanisms. We are going to study the concurrency control mechanisms in detail in tomorrow's lecture.