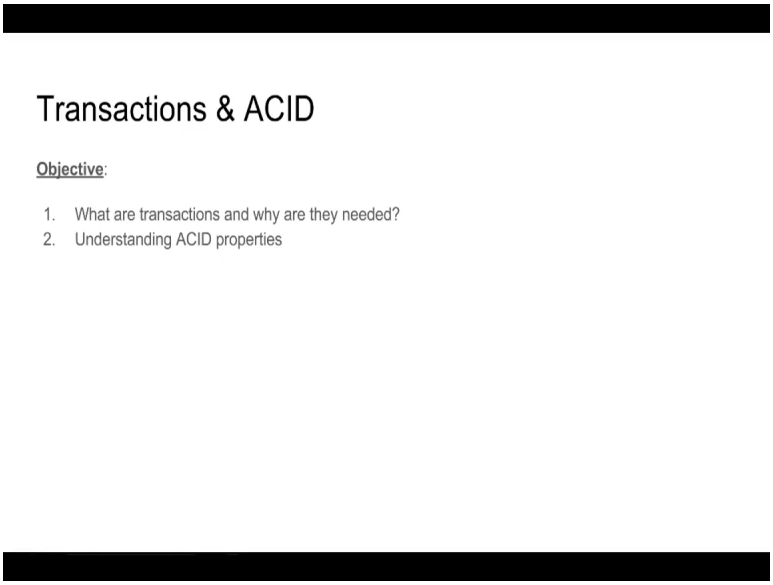
Introduction to Morden Application Development
Dr. Gaurav Raina
Prof. Tanmai Gopal
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

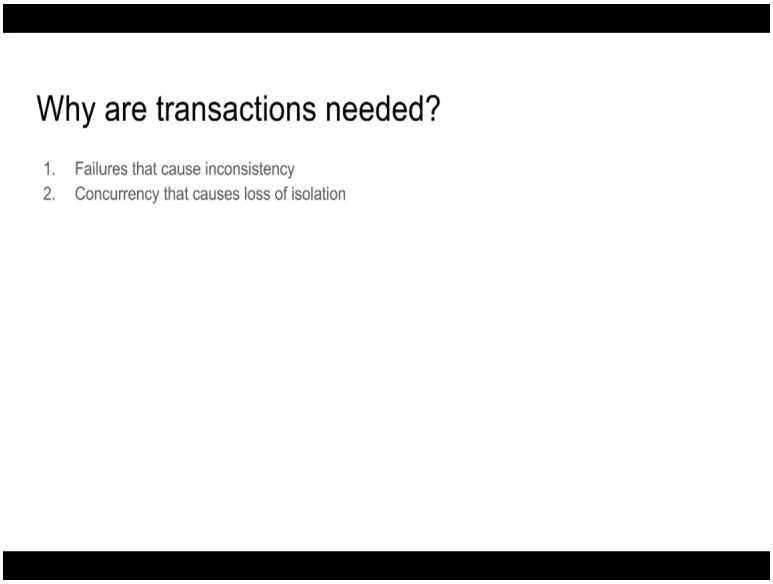
Module – 09
Lecture – 18
Transactions & ACID properties

(Refer Slide Time: 00:04)



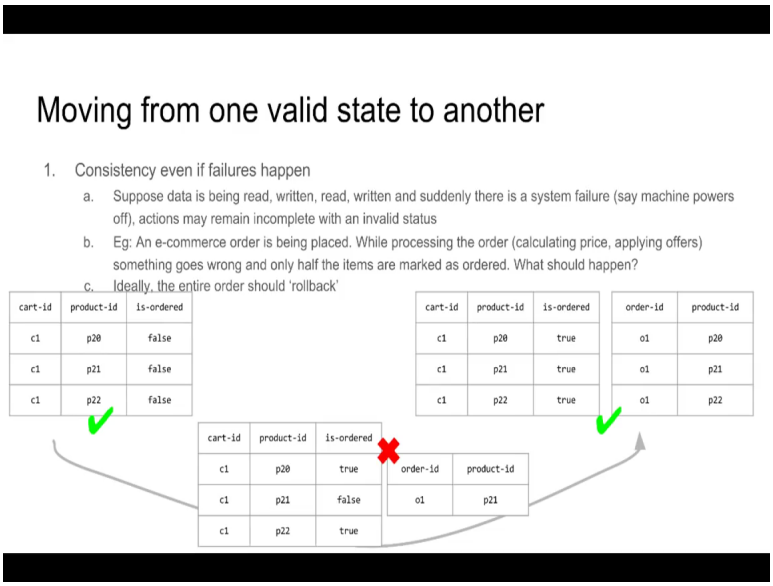
Hi all, welcome to module 9. In this module we will be talking about transactions and ACID properties. Our 2 objectives are to understand why transactions are needed and what ACID properties mean.

(Refer Slide Time: 00:12)



So, we are going to look at 2 specific use cases to understand why transactions are needed, we are going to look at failures and the system that can cause inconsistency in our database and we are also going to look at concurrency and how concurrency can cause a loss of isolation. Do not worry about what these mean because we will be understanding that VR examples over the course of this module.

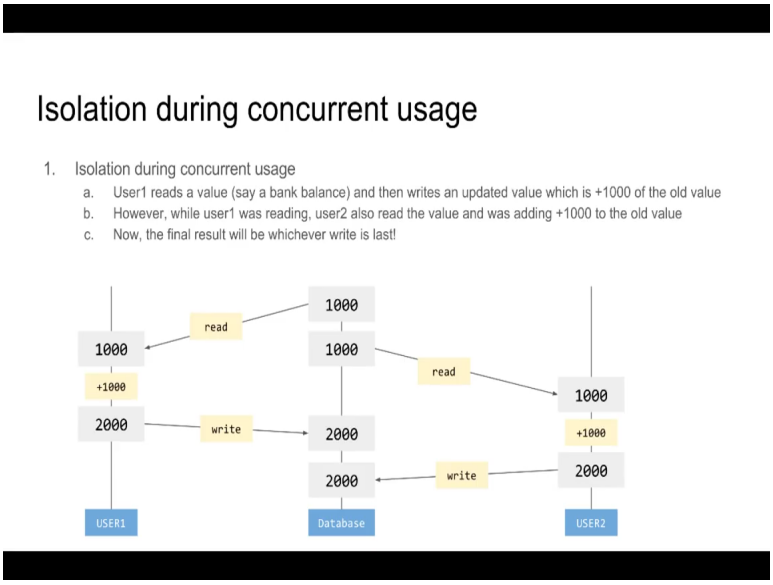
(Refer Slide Time: 00:39)



So, first let us try to understand that how failures can cause invalid states to exist in our database suppose we are reading data and writing data to a database and suddenly that is the system failure which could be a power failure or a disc failure or a software failure and this causes our database to an to be in an invalid state. So, just to take an example of an E-commerce kind of modelling on the left we have a shopping cart and the shopping cart contains three products and none of these and this entire shopping cart has not been ordered. So, we see that is ordered flag is set to false. Once the entire order is placed and the payment is done let us say we move to another state of a database where all the is ordered flags are turned to true and in the ordered table three new rows will be created let us say that the order is been placed for three of these orders.

Now, while both the state on the left and the state on the right are completely valid states because of the system failure in the middle may be all of the items in the tables might not get updated and we might end up with the state like this where not all the is ordered values are updated and all the new rows have not been created. So, this is an invalid state right. Ideally what we want is we want to be here or we want to be here we never want to have a situation where we are in the middle and if by chance there is a failure then we should be able to roll back to the previous state and then we should be able to restart the entire order process again this process of undoing our operations is called rolling back. So, this is one use case where we need some kind of. So, this is an example of where we need some kind of atomicity to ensure that our database is in a valid state and by atomicity we mean that we should have an all or none kind of situation where either the entire set of operations happen or they do not happen at all.

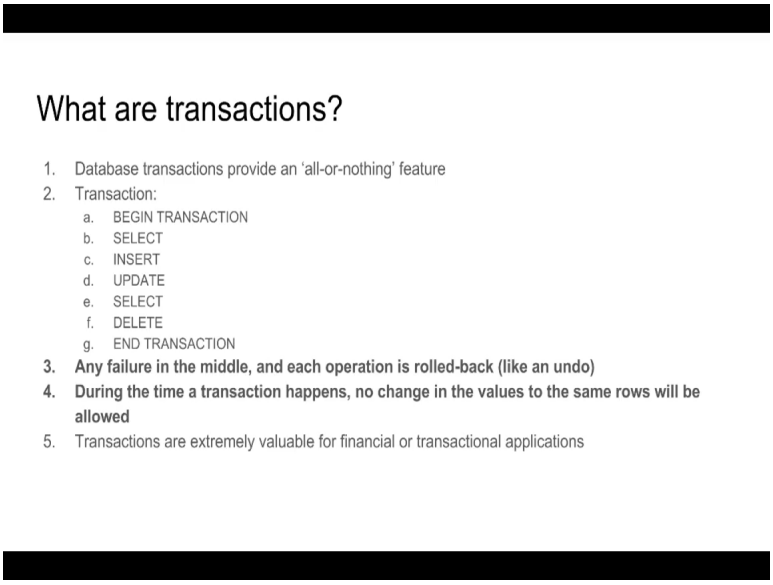
(Refer Slide Time: 02:22)



Let us look at another example slightly more complex example let us say we have a database which contains table which contains a value let us say which is a bank balance right. So, we have a 1000 rupee back balance to start with. Now let us say somebody is going to update this bank balance let us say somebody is adding a 1000 rupees to our bank accounts. So, user 1 is reading the value 1000 is in adding the value plus 1000 to it and is writing the new value. So, the new value 2000 gets written. But while user 1 is reading and witting this new value user 2 is also trying to add a 1000 rupees to this bank balance and let us say 1000 rupees is being read 1000 rupees was added and 2000 rupees is 2 things that the new balance is 2000 rupees and tries to write this new balance of 2000 rupees.

So, what is happening is that before the first write came through the second user already read and a value which was actually invalid right. So, we want to prevent this some happening while this is going on and this business of desiring that user 1's actions are different from user 2's action is called isolation. So, we want these operations to be in isolation from these operations. So, this is another use case of why we need something special in the database to be able to handle these use cases.

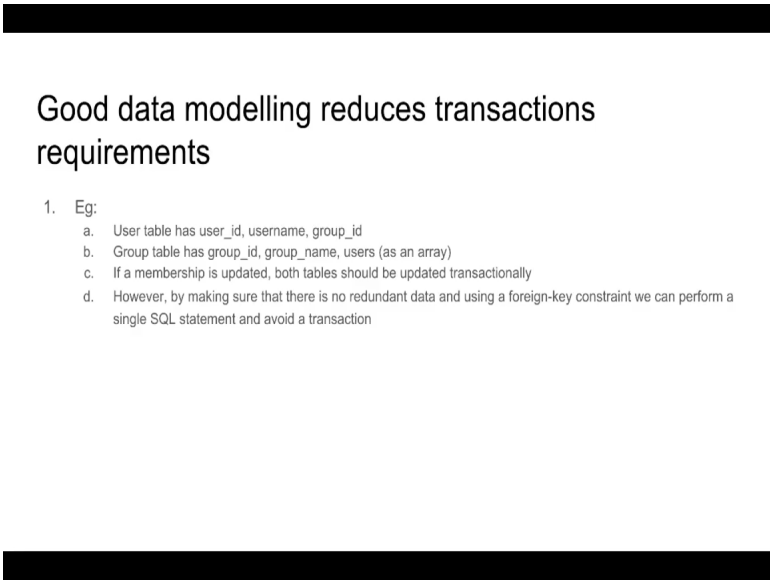
(Refer Slide Time: 03:40).



This is where database transactions come in. They provide in essence all are nothing kind of future and they also provide this kind of an isolation that we need for example, instead of issuing SQL statements one by one, we would issue an SQL statement called begin tradition after which might be we will do a select operation and insert operation, update operation, delete operation we do a bunch of other SQL operations and after finishing our SQL operations we issue last statement called end transaction.

Now, this entire sequence of steps is called a transaction in case that is the system failure or something goes wrong the database will ensure that the entire that all of these operations are rolled back and we see the database as it was before this, begin transaction happened. So, this is how we handle failures with transactions. Suppose multiple transactions that happening right like in this case where user 1 has a transaction and user 2 has a transaction if user 1's transactions and user 2's transactions cause a conflict by updating or referring to the same value one of the transactions will be deliberately failed and will be rolled back so that we have an isolation effect between these 2 transactions. So, as you can imagine transactions are extremely valuable for financial or transactional applications.

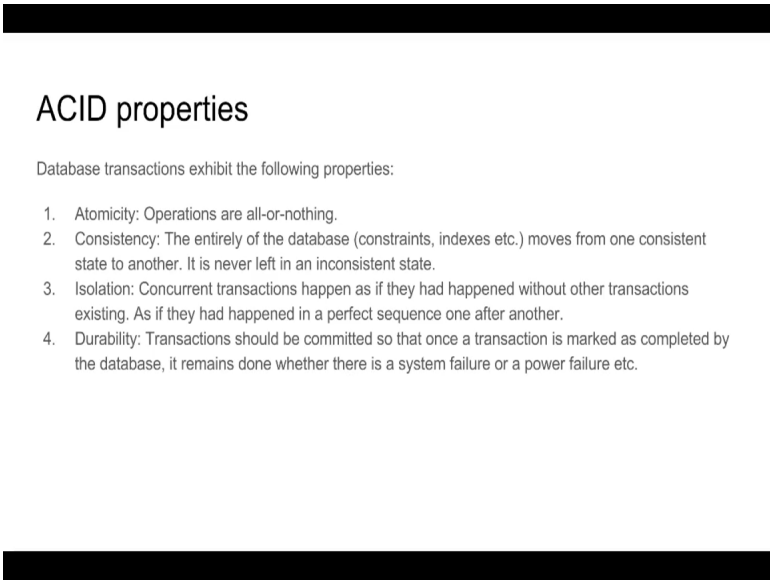
(Refer Slide Time: 04:50)



A lot of the times modelling a data well and specially normalising a data models it can reduce the requirements for having transaction let us say for example, we have a simple user and groups kind of modelling. So, we have a user table and a groups table inside the user table we store the user id the user name and the group that the user belongs to right and in the group table let us say that we store the group id the group name and then array of users say as JSON column. So, in this situation if membership is updated let us say the group of a particular user is changed not only do we have to obtain the user table and change the group ID, but we also have to go to the group table and update the users array we have to do both of these operations transactionally. If we do one operation and the other operation does not go through we will have an inconsistent state and we will not know where the truth is we will not know whether the group table has the true data or the user table has the true data.

So, modelling a data well and ensuring that this user's array is not used and we had only having and we are only using a group id here which has a foreign key constraint to the group table this will ensure that we do not actually need to use a transaction and we can get away with this single SQL statement. These properties of transactions that we have talked about are actually called ACID properties.

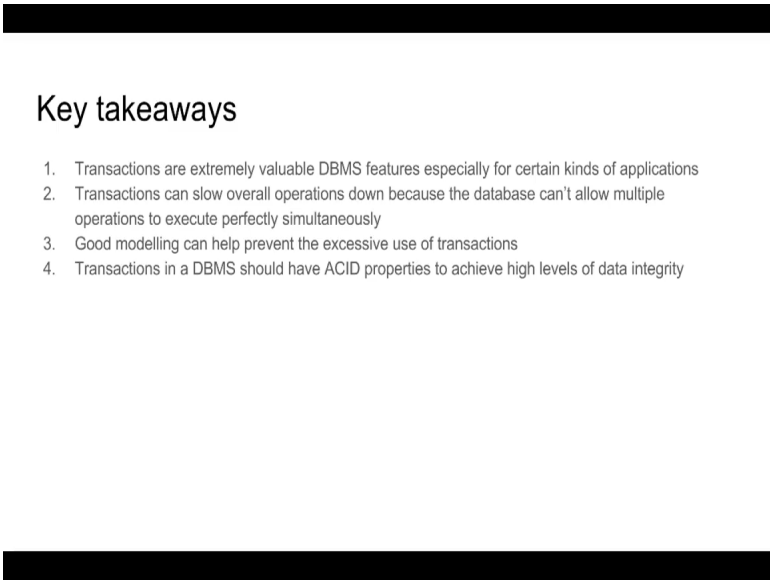
(Refer Slide Time: 05:54)



So, let us understand what ACID stands for. Atomicity which means that operations are all or nothing consistency means that the entire database which means that if we have foreign key constraints or indexes all the structures in the database move from one consistent state to an another consistent state. Isolation refers to the property that while concurrent transactions that happening they are perfectly equivalent to those transactions having executed in sequence one after another.

Durability refers to the fact that transaction should actually be committed to disk. So, that if a transaction is marked as completed it remains safe to the disk even if the database is restarted or if there is any kind of failure that causes the database to restart. So, transactions that have ACID properties are very important to creating applications that are stable and need a very high amount of data integrity.

(Refer Slide Time: 06:46)



The most important things you take away from this module are the transactions are very valuable DBMS features especially for certain kinds of applications. It is important to understand that transactions do not have to be provided by the database we can create transactions in the application as well, but we would have to write much more complicated application logic to ensure that while a particular transaction is happening no other transactions are allowed we would have to write application logic to ensure that in case there is a failure variable to recover from those failures. So, that operations can be rolled back and we would essentially have to do all of the work that DBMS is doing for us in our application code.

The price that we pay for transactions is that it can slow over all operations down because the database cannot do multiple operations perfectly concurrently we also understood that doing good data modelling can prevent the excessive use of transactions and we know that a transaction to have a high degree of data integrity must support all 4 ACID properties.