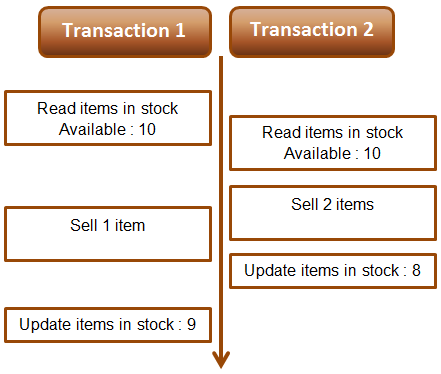
In this video we will discuss, **lost update problem in sql server** with an example.   
  
   
  
**Lost update problem happens when 2 transactions read and update the same data**. Let's understand this with an example. We will use the following table **tblInventory**for this example.   
  
lost update concurrency problem example   
  
As you can see in the diagram below there are 2 transactions - Transaction 1 and Transaction 2. Transaction 1 starts first, and it is processing an order for 1 iPhone. It sees ItemsInStock as 10.   
  
At this time Transaction 2 is processing another order for 2 iPhones. It also sees ItemsInStock as 10. Transaction 2 makes the sale first and updates ItemsInStock with a value of 8.   
  
At this point Transaction 1 completes the sale and silently overwrites the update of Transaction 2. As Transaction 1 sold 1 iPhone it has updated ItemsInStock to 9, while it actually should have updated it to 7.   
  
   
  
**Example :** The lost update problem example. Open 2 instances of SQL Server Management studio. From the first window execute Transaction 1 code and from the second window, execute Transaction 2 code. Transaction 1 is processing an order for 1 iPhone, while Transaction 2 is processing an order for 2 iPhones. At the end of both the transactions ItemsInStock must be 7, but we have a value of 9. This is because Transaction 1 silently overwrites the update of Transaction 2. This is called the **lost update problem**. 

-- Transaction 1

Begin Tran

Declare @ItemsInStock int

Select @ItemsInStock = ItemsInStock

from tblInventory where Id=1

-- Transaction takes 10 seconds

Waitfor Delay '00:00:10'

Set @ItemsInStock = @ItemsInStock - 1

Update tblInventory

Set ItemsInStock = @ItemsInStock where Id=1

Print @ItemsInStock

Commit Transaction

-- Transaction 2

Begin Tran

Declare @ItemsInStock int

Select @ItemsInStock = ItemsInStock

from tblInventory where Id=1

-- Transaction takes 1 second

Waitfor Delay '00:00:1'

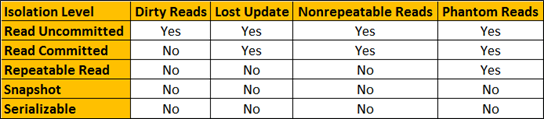
Set @ItemsInStock = @ItemsInStock - 2

Update tblInventory

Set ItemsInStock = @ItemsInStock where Id=1

Print @ItemsInStock

Commit Transaction

Both Read Uncommitted and Read Committed transaction isolation levels have the lost update side effect. Repeatable Read, Snapshot, and Serializable isolation levels does not have this side effect. If you run the above Transactions using any of the higher isolation levels (Repeatable Read, Snapshot, or Serializable) you will not have lost update problem. The repeatable read isolation level uses additional locking on rows that are read by the current transaction, and prevents them from being updated or deleted elsewhere. This solves the lost update problem.   
  
   
  
For both the above transactions, set Repeatable Read Isolation Level. Run Transaction 1 first and then a few seconds later run Transaction 2. Transaction 1 completes successfully, but Transaction 2 competes with the following error.  
Transaction was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Once you rerun Transaction 2, ItemsInStock will be updated correctly as expected.