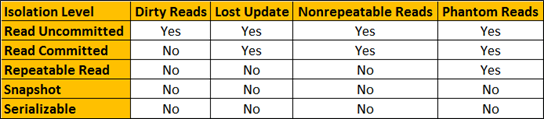
In this video we will discuss, **snapshot isolation level in sql server** with examples.   
  
   
  
As you can see from the table below, just like serializable isolation level, snapshot isolation level does not have any concurrency side effects.   
   
  
**What is the difference between serializable and snapshot isolation levels**  
Serializable isolation is implemented by acquiring locks which means the resources are locked for the duration of the current transaction. This isolation level does not have any concurrency side effects but at the cost of significant reduction in concurrency.   
  
Snapshot isolation doesn't acquire locks, it maintains versioning in Tempdb. Since, snapshot isolation does not lock resources, it can significantly increase the number of concurrent transactions while providing the same level of data consistency as serializable isolation does.  
  
Let us understand Snapshot isolation with an example. We will be using the following table tblInventory for this example.   
snapshot isolation 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. Notice that Transaction 2 is blocked until Transaction 1 is completed. 

--Transaction 1

Set transaction isolation level serializable

Begin Transaction

Update tblInventory set ItemsInStock = 5 where Id = 1

waitfor delay '00:00:10'

Commit Transaction

-- Transaction 2

Set transaction isolation level serializable

Select ItemsInStock from tblInventory where Id = 1

Now change the isolation level of Transaction 2 to snapshot. To set snapshot isolation level, it must first be enabled at the database level, and then set the transaction isolation level to snapshot. 

-- Transaction 2

-- Enable snapshot isloation for the database

Alter database SampleDB SET ALLOW\_SNAPSHOT\_ISOLATION ON

-- Set the transaction isolation level to snapshot

Set transaction isolation level snapshot

Select ItemsInStock from tblInventory where Id = 1

From the first window execute Transaction 1 code and from the second window, execute Transaction 2 code. Notice that Transaction 2 is not blocked and returns the data from the database as it was before Transaction 1 has started.   
  
**Modifying data with snapshot isolation level :** Now let's look at an example of what happens when a transaction that is using snapshot isolation tries to update the same data that another transaction is updating at the same time.   
  
In the following example, Transaction 1 starts first and it is updating ItemsInStock to 5. At the same time, Transaction 2 that is using snapshot isolation level is also updating the same data. Notice that Transaction 2 is blocked until Transaction 1 completes. When Transaction 1 completes, Transaction 2 fails with the following error to prevent concurrency side effect - Lost update. If Transaction 2 was allowed to continue, it would have changed the ItemsInStock value to 8 and when Transaction 1 completes it overwrites ItemsInStock to 5, which means we have lost an update. To complete the work that Transaction 2 is doing we will have to rerun the transaction.   
  
Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.tblInventory' directly or indirectly in database 'SampleDB' to update, delete, or insert the row that has been modified or deleted by another transaction. Retry the transaction or change the isolation level for the update/delete statement.

--Transaction 1

Set transaction isolation level serializable

Begin Transaction

Update tblInventory set ItemsInStock = 5 where Id = 1

waitfor delay '00:00:10'

Commit Transaction

-- Transaction 2

-- Enable snapshot isloation for the database

Alter database SampleDB SET ALLOW\_SNAPSHOT\_ISOLATION ON

-- Set the transaction isolation level to snapshot

Set transaction isolation level snapshot

Update tblInventory set ItemsInStock = 8 where Id = 1