**In this video we will discuss**   
1. How SQL Server detects deadlocks  
2. What happens when a deadlock is detected  
3. What is DEADLOCK\_PRIORITY  
4. What is the criteria that SQL Server uses to choose a deadlock victim when there is a deadlock   
  
   
  
This is continuation to [Part 78](http://csharp-video-tutorials.blogspot.com/2015/08/sql-server-deadlock-example.html), please watch [Part 78](http://csharp-video-tutorials.blogspot.com/2015/08/sql-server-deadlock-example.html) before proceeding.   
  
**How SQL Server detects deadlocks**  
Lock monitor thread in SQL Server, runs every 5 seconds by default to detect if there are any deadlocks. If the lock monitor thread finds deadlocks, the deadlock detection interval will drop from 5 seconds to as low as 100 milliseconds depending on the frequency of deadlocks. If the lock monitor thread stops finding deadlocks, the Database Engine increases the intervals between searches to 5 seconds.   
  
**What happens when a deadlock is detected**  
When a deadlock is detected, the Database Engine ends the deadlock by choosing one of the threads as the deadlock victim. The deadlock victim's transaction is then rolled back and returns a 1205 error to the application. Rolling back the transaction of the deadlock victim releases all locks held by that transaction. This allows the other transactions to become unblocked and move forward.   
  
**What is DEADLOCK\_PRIORITY**  
By default, SQL Server chooses a transaction as the deadlock victim that is least expensive to roll back. However, a user can specify the priority of sessions in a deadlock situation using the SET DEADLOCK\_PRIORITY statement. The session with the lowest deadlock priority is chosen as the deadlock victim.   
  
Example : SET DEADLOCK\_PRIORITY NORMAL   
  
**DEADLOCK\_PRIORITY**  
1. The default is Normal  
2. Can be set to LOW, NORMAL, or HIGH  
3. Can also be set to a integer value in the range of -10 to 10.  
 LOW : -5  
 NORMAL : 0  
 HIGH : 5   
  
**What is the deadlock victim selection criteria**  
1. If the DEADLOCK\_PRIORITY is different, the session with the lowest priority is selected as the victim  
2. If both the sessions have the same priority, the transaction that is least expensive to rollback is selected as the victim  
3. If both the sessions have the same deadlock priority and the same cost, a victim is chosen randomly   
  
**SQL Script to setup the tables for the examples** 

Create table TableA

(

    Id int identity primary key,

    Name nvarchar(50)

)

Go

Insert into TableA values ('Mark')

Insert into TableA values ('Ben')

Insert into TableA values ('Todd')

Insert into TableA values ('Pam')

Insert into TableA values ('Sara')

Go

Create table TableB

(

    Id int identity primary key,

    Name nvarchar(50)

)

Go

Insert into TableB values ('Mary')

Go

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. We have not explicitly set DEADLOCK\_PRIORITY, so both the sessions have the default DEADLOCK\_PRIORITY which is NORMAL. So in this case SQL Server is going to choose Transaction 2 as the deadlock victim as it is the least expensive one to rollback.

-- Transaction 1

Begin Tran

Update TableA Set Name = Name + ' Transaction 1' where Id IN (1, 2, 3, 4, 5)

-- From Transaction 2 window execute the first update statement

Update TableB Set Name = Name + ' Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement

Commit Transaction

-- Transaction 2

Begin Tran

Update TableB Set Name = Name + ' Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

Update TableA Set Name = Name + ' Transaction 2' where Id IN (1, 2, 3, 4, 5)

-- After a few seconds notice that this transaction will be chosen as the deadlock

-- victim as it is less expensive to rollback this transaction than Transaction 1

Commit Transaction

In the following example we have set DEADLOCK\_PRIORITY of Transaction 2 to HIGH. Transaction 1 will be chosen as the deadlock victim, because it's DEADLOCK\_PRIORITY (Normal) is lower than the DEADLOCK\_PRIORITY of Transaction 2. 

-- Transaction 1

Begin Tran

Update TableA Set Name = Name + ' Transaction 1' where Id IN (1, 2, 3, 4, 5)

-- From Transaction 2 window execute the first update statement

Update TableB Set Name = Name + ' Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement

Commit Transaction

-- Transaction 2

SET DEADLOCK\_PRIORITY HIGH

GO

Begin Tran

Update TableB Set Name = Name + ' Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

Update TableA Set Name = Name + ' Transaction 2' where Id IN (1, 2, 3, 4, 5)

-- After a few seconds notice that Transaction 2 will be chosen as the

-- deadlock victim as it's DEADLOCK\_PRIORITY (Normal) is lower than the

-- DEADLOCK\_PRIORITY this transaction (HIGH)

Commit Transaction