/****************/
/* SQL SERVER   */
/*   INDEX   */
/****************/

--72-- CONCURRENT TRANSACTION PROBLEMS: LOST UPDATE WITH READ UNCOMMITTED AND READ COMMITTED
--73-- CONCURRENT TRANSACTION PROBLEMS: NON REPEATABLE READ WITH READ UNCOMMITED AND READ COMMITED
--74-- CONCURRENT TRANSACTION PROBLEMS: PHANTOM READ WITH READ UNCOMMITTED, READ COMMITED, AND REPEATABLE READ
--75-- CONCURRENT TRANSACTION PROBLEMS: SNAPSHOT ISOLATION LEVEL VS SERIALIZABLE ISOLATION LEVEL
--76-- READ COMMITED SNAPSHOT ISOLATION LEVEL
--77-- READ COMMITED SNAPSHOT ISOLATION LEVEL VS SNAPSHOT ISOLATION LEVEL
--78-- DEADLOCK EXAMPLE
--79-- DEADLOCK VICTIM SELECTION
--80-- LOGGING DEADLOCKS
--81-- DEADLOCK ANALYSIS AND PREVENTION
--82-- CAPTURING DEADLOCKS IN SQL PROFILER
--83-- SQL SERVER DEADLOCK ERROR HANDLING
--84-- HANDLING DEADLOCKS IN ADO.NET
--85-- RETRY LOGIC FOR DEADLOCK EXCEPTIONS
--86-- HOW TO FIND BLOCKING QUERIES IN SQL SERVER
--87-- SQL SERVER EXCEPT OPERATOR
--88-- DIFFERENCE BETWEEN EXCEPT AND NOT IN SQL SERVER
--89-- INTERSECT OPERATOR IN SQL SERVER
--90-- DIFFERENCE BETWEEN UNION INTERESECT AND EXCEPT IN SQL SERVER
--91-- CROSS APPLY AND OUTER APPLY IN SQL SERVER
--92-- DDL TRIGGERS IN SQL SERVER
--93-- SERVER SCOPED DDL TRIGGERS
--94-- SQL SERVER TRIGGER EXECUTION ORDER
--95-- AUDIT TABLE CHANGES IN SQL SERVER

/**************/
/* SQL SERVER */
/**************/

/
****
****
*72****** CONCURRENT TRANSACTION PROBLEMS: LOST UPDATE WITH READ UNCOMMITTED AND READ COMMITTED **********72*
****
*****************************/

-- 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.

-- the lost update problem example

-- 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'
-- make a sale of 1 item
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 '00:00:01'}
-- make a sale of 2 items (before transaction 1)
Set @ItemsInStock = @ItemsInStock - 2

Update tblInventory
Set ItemsInStock = @ItemsInStock where Id=1

Print @ItemsInStock

Commit Transaction

-- execute transaction 1 and 2 and 2 finishes first, it will print 8, but first transaction will silently
overwrite
-- the sales of transaction 2.

-- 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.

-- sql server transaction isolation levels

-- 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.



/
**********************************************************************************************
**********
```

*73****** CONCURRENT TRANSACTION PROBLEMS: NON REPEATABLE READ WITH READ UNCOMMITED AND READ COMMITED ***********73*
**********/

-- Non repeatable read problem happens when one transaction reads the same data twice and another transaction
--updates that data in between the first and second read of transaction one.

-- We will use the following table tblInventory in this demo
-- sql server non-repeatable read

-- The following diagram explains the problem : Transaction 1 starts first. Reads ItemsInStock. Gets a value
-- of 10 for first read. Transaction 1 is doing some work and at this point Transaction 2 starts and
-- UpdatesItemsInStock to 5. Transaction 1 then makes a second read. At this point Transaction 1 gets a
-- value of 5, reulting in non-repeatable read problem.

-- non repeatable read example in sql server

-- Non-repeatable read 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 when Transaction 1
-- completes, it gets different values for read 1 and read 2, resulting in non-repeatable read.

-- Transaction 1
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1

-- Do Some work
waitfor delay '00:00:10'

Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

-- Transaction 2
Update tblInventory set ItemsInStock = 5 where Id = 1

-- Repeatable read or any other higher isolation level should solve the non-repeatable read problem.
-- sql server transaction isolation levels

-- Fixing non repeatable read concurrency problem : To fix the non-repeatable read problem, set transaction
-- isolation level of Transaction 1 to repeatable read. This will ensure that the data that Transaction 1 has read,
-- will be prevented from being updated or deleted elsewhere. This solves the non-repeatable read problem.

```
-- When you execute Transaction 1 and 2 from 2 different instances of SQL Server management studio, Transaction
-- 2 is blocked until Transaction 1 completes and at the end of Transaction 1, both the reads get the same value for ItemsInStock.

-- Transaction 1
Set transaction isolation level repeatable read
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1

-- Do Some work
waitfor delay '00:00:10'

Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

-- Transaction 2
Update tblInventory set ItemsInStock = 5 where Id = 1



/
**********************************************************************************************************************
*74****** CONCURRENT TRANSACTION PROBLEMS: PHANTOM READ WITH READ UNCOMMITTED, READ COMMITED, AND REPEATABLE READ **********74*
**********************************************************************************************************************/

-- Phantom read happens when one transaction executes a query twice and it gets a different number of rows in the result set each time.
-- This happens when a second transaction inserts a new row that matches the WHERE clause of the query executed by the first transaction.

-- We will use the following table tblEmployees in this demo
-- phantom reads in sql server

-- ScripT to create the table tblEmployees
Create table tblEmployees
(
   Id int primary key,
   Name nvarchar(50)
)
Go

Insert into tblEmployees values(1,'Mark')
Insert into tblEmployees values(3, 'Sara')
Insert into tblEmployees values(100, 'Mary')
```

-- The following diagram explains the problem : Transaction 1 starts first. Reads from Emp table where Id between 1 and 3.
-- 2 rows retrieved for first read. Transaction 1 is doing some work and at this point Transaction 2 starts and inserts a new
-- employee with Id = 2. Transaction 1 then makes a second read. 3 rows retrieved for second read, reulting in phantom read problem.

-- phantom reads example in sql server

-- Phantom read 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 when Transaction 1 completes, it gets different number of rows for
-- read 1 and read 2, resulting in phantom read.

```
-- Transaction 1
Begin Transaction
Select * from tblEmployees where Id between 1 and 3
-- Do Some work
waitfor delay '00:00:10'
Select * from tblEmployees where Id between 1 and 3
Commit Transaction
```

```
-- Transaction 2
Insert into tblEmployees values(2, 'Marcus')
```

-- Serializable or any other higher isolation level should solve the phantom read problem.
-- sql server transaction isolation levels

-- Fixing phantom read concurrency problem : To fix the phantom read problem, set transaction isolation level of Transaction 1
-- to serializable. This will place a range lock on the rows between 1 and 3, which prevents any other transaction from
-- inserting new rows with in that range. This solves the phantom read problem.

-- When you execute Transaction 1 and 2 from 2 different instances of SQL Server management studio, Transaction 2 is blocked
-- until Transaction 1 completes and at the end of Transaction 1, both the reads get the same number of rows.

-- first let´s restore previous settings to repeat the simulation of the problem

```
Delete from tblEmployees where Id = 2
```

```
-- Transaction 1
Set transaction isolation level serializable
Begin Transaction
Select * from tblEmployees where Id between 1 and 3
-- Do Some work
waitfor delay '00:00:10'
```

```
Select * from tblEmployees where Id between 1 and 3
Commit Transaction

-- Transaction 2

Insert into tblEmployees values(2, 'Marcus')


-- Transaction 2 will be allowed to continue only after Transaction 1 has finished.
-- This is because with serializable, there has been a locked placed specifically between rows 1 and 3

-- Difference between repeatable read and serializable
-- Repeatable read prevents only non-repeatable read. Repeatable read isolation level ensures that the data that one transaction has
-- read, will be prevented from being updated or deleted by any other transaction, but it doe not prevent new rows from being
-- inserted by other transactions resulting in phantom read concurrency problem.

-- Serializable prevents both non-repeatable read and phantom read problems. Serializable isolation level ensures that the data that
-- one transaction has read, will be prevented from being updated or deleted by any other transaction. It also prevents new rows
-- from being inserted by other transactions, so this isolation level prevents both non-repeatable read and phantom read problems.


/
**************************************************************************************************************************
*75****** CONCURRENT TRANSACTION PROBLEMS: SNAPSHOT ISOLATION LEVEL VS SERIALIZABLE ISOLATION LEVEL **********75*
**************************************************************************************************************************/

-- As you can see from the table below, just like serializable isolation level, snapshot isolation level does not have any concurrency side effects.
-- sql server transaction isolation levels

-- 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.

-- Error Message
-- 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

-- Read committed snapshot isolation level is not a different isolation level. It is a different way of implementing Read committed isolation level.
-- One problem we have with Read Committed isloation level is that, it blocks the transaction if it is trying to read the data,
-- that another transaction is updating at the same time.

-- The following example demonstrates the above point. 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 Read Committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

```
-- We can make Transaction 2 to use row versioning technique instead of locks by enabling Read committed snapshot isolation
-- at the database level. Use the following command
-- to enable READ_COMMITTED_SNAPSHOT isolation
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT ON

-- Please note : For the above statement to execute successfully all the other database connections should be closed.

-- After enabling READ_COMMITTED_SNAPSHOT, execute Transaction 1 first and then Transaction 2 simultaneously.
-- Notice that the Transaction 2 is not blocked. It immediately returns the committed data that is in the database
-- before Transaction 1 started. This is because Transaction 2 is now using Read committed snapshot isolation level.

-- Let's see if we can achieve the same thing using snapshot isolation level instead of read committed snapshot isolation level.

-- Step 1 : Turn off READ_COMMITTED_SNAPSHOT
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT OFF

-- Step 2 : Enable snapshot isolation level at the database level
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION ON

-- Step 3 : Execute Transaction 1 first and then Transaction 2 simultaneously. Just like in the previous example,
-- notice that the Transaction 2 is not blocked. It immediately returns the committed data that is in the database
-- before Transaction 1 started.

--Transaction 1
Set transaction isolation level Read Committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level snapshot
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

-- So what is the point in using read committed snapshot isolation level over snapshot isolation level?
-- There are some differences between read committed snapshot isolation level and snapshot isolation level.
-- We will discuss these in our next video.
```

```
/
*****************************************************************************
*76****************** READ COMMITED SNAPSHOT ISOLATION LEVEL
*******************76*
*****************************************************************************
*/

-- In this video we will discuss Read committed snapshot isolation level in sql server. This is
continuation Part 75.
-- Please watch Part 75 from SQL Server tutorial before proceeding.

-- We will use the following table tblInventory in this demo

-- Read committed snapshot isolation level example
-- Read committed snapshot isolation level is not a different isolation level. It is a different way of
implementing Read committed isolation level.
-- One problem we have with Read Committed isloation level is that, it blocks the transaction if it is
trying to read the data,
-- that another transaction is updating at the same time.

-- The following example demonstrates the above point. 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 Read Committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

-- We can make Transaction 2 to use row versioning technique instead of locks by enabling Read
committed snapshot isolation
-- at the database level. Use the following command
-- to enable READ_COMMITTED_SNAPSHOT isolation
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT ON

-- Please note : For the above statement to execute successfully all the other database connections
should be closed.
```

```
-- After enabling READ_COMMITTED_SNAPSHOT, execute Transaction 1 first and then Transaction 2 simultaneously.
-- Notice that the Transaction 2 is not blocked. It immediately returns the committed data that is in the database
-- before Transaction 1 started. This is because Transaction 2 is now using Read committed snapshot isolation level.

-- Let's see if we can achieve the same thing using snapshot isolation level instead of read committed snapshot isolation level.

-- Step 1 : Turn off READ_COMMITTED_SNAPSHOT
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT OFF

-- Step 2 : Enable snapshot isolation level at the database level
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION ON

-- Step 3 : Execute Transaction 1 first and then Transaction 2 simultaneously. Just like in the previous example,
-- notice that the Transaction 2 is not blocked. It immediately returns the committed data that is in the database
-- before Transaction 1 started.

--Transaction 1
Set transaction isolation level Read Committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level snapshot
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1
Commit Transaction

-- So what is the point in using read committed snapshot isolation level over snapshot isolation level?
-- There are some differences between read committed snapshot isolation level and snapshot isolation level.
-- We will discuss these in our next video.



/
**********************************************************************************
*77***** READ COMMITED SNAPSHOT ISOLATION LEVEL VS SNAPSHOT ISOLATION LEVEL *****77*
**********************************************************************************
*/
```

-- In this video we will discuss the differences between snapshot isolation and read committed snapshot
-- isolation in sql server. This is continuation to Parts 75 and 76. Please watch Part 75 and 76 from
-- SQL Server tutorial before proceeding.

-- Read Committed Snapshot isolation
Snapshot Isolation
-- No update conflicts
Vulnerable to update conflicts
-- Works with existing applications without requiring any change to the application
Application change may be required to use with an existing application
-- Can be used with distributed transactions
Cannot be used with distributed transactions
-- Provides statement-level read consistency
Provides transaction-level read consistency

-- Update conflicts : Snapshot isolation is vulnerable to update conflicts where as Read Committed Snapshot
-- Isolation is not. When a transaction running under snapshot isolation tries to update data that another
-- transaction is already updating at the sametime, an update conflict occurs and the transaction terminates and rolls back with an error.

--We will use the following table tblInventory in this demo

--Read committed snapshot isolation level example

--Enable Snapshot Isolation for the SampleDB database using the following command
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION ON

--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.
-- When Transaction 1 completes, Transaction 2 raises an update conflict and the transaction terminates and rolls back with an error.

--Transaction 1
Set transaction isolation level snapshot
Begin Transaction
Update tblInventory set ItemsInStock = 8 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level snapshot
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
Commit Transaction

-- Now let's try the same thing using Read Committed Sanpshot Isolation

-- Step 1 : Disable Snapshot Isolation for the SampleDB database using the following command
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION OFF

-- Step 2 : Enable Read Committed Sanpshot Isolation at the database level using the following command
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT ON

-- Step 3 : 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. When Transaction 1 completes, Transaction 2 also completes successfully without any update conflict.

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
Update tblInventory set ItemsInStock = 8 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
Commit Transaction

-- Existing application : If your application is using the default Read Committed isolation level, you can very
-- easily make the application to use Read Committed Snapshot Isolation without requiring any change to the
-- application at all. All you need to do is turn on READ_COMMITTED_SNAPSHOT option in the database, which will
-- change read committed isolation to use row versioning when reading the committed data.

-- Distributed transactions : Read Committed Snapshot Isolation works with distributed transactions, whereas snapshot isolation does not.

-- Read consistency : Read Committed Snapshot Isolation provides statement-level read consistency where as Snapshot
-- Isolation provides transaction-level read consistency. The following diagrams explain this.

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
Update tblInventory set ItemsInStock = 8 where Id = 1
waitfor delay '00:00:10'

Commit Transaction

-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
Select * from tblInventory where Id = 1
-- breakpoint
Select * from tblInventory where Id = 1
Commit Transaction

-- Suppose at an initial state ItemsInStock = 10

-- Then Suppose we start transaction 1 but while its processing we start the first part of transaction 2 (until the breakpoint)
-- We'll get a return value from the Select statement of 10, suppose now the first transaction finishes, and we run
-- the second part of transaction 2.

-- If we are at a read committed snapshot isolation level, we´ll get 10 in the first select, and 8 in the second
-- If we are at a snapshot isolation level, we'll get 10 in the first select and 10 in the second.

-- This is because the read committed snapshot isolation will return the last commited data before the select statement
-- began (in the middle of the transaction 2) and not the last committed data before the transaction 2 began.
-- Snapshot isolation however will return the last committed data before the transaction 2 began on both select staments.

/**
*78****************** DEADLOCK EXAMPLE ********************78*
**/

-- When can a deadlock occur
-- In a database, a deadlock occurs when two or more processes have a resource locked, and each process requests
-- a lock on the resource that another process has already locked. Neither of the transactions here can move
-- forward, as each one is waiting for the other to release the lock. The following diagram explains this.

-- SQL Server deadlock example

-- When deadlocks occur, SQL Server will choose one of processes as the deadlock victim and rollback that
-- process, so the other process can move forward. The transaction that is chosen as the deadlock victim
-- will produce the following error.

Msg 1205, Level 13, State 51, Line 1

Transaction (Process ID 57) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

-- Let us look at this in action. We will use the following 2 tables for this example.

SQL script to create the tables and populate them with test data

```
Create table TableA
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableA values ('Mark')
Go

Create table TableB
(
   Id int identity primary key,
   Name nvarchar(50)
)
Go

Insert into TableB values ('Mary')

Go
```

-- The following 2 transactions will result in a dead lock. 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
-- EXECUTE STEP 1)
Begin Tran
Update TableA Set Name = 'Mark Transaction 1' where Id = 1

-- From Transaction 2 window execute the first update statement

-- EXECUTE STEP 3)
Update TableB Set Name = 'Mary Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement
Commit Transaction
```

```
-- Transaction 2
-- EXECUTE STEP 2)
Begin Tran
Update TableB Set Name = 'Mark Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

-- EXECUTE STEP 4)
Update TableA Set Name = 'Mary Transaction 2' where Id = 1


-- STEP 5)
-- After a few seconds notice that one of the transactions complete
-- successfully while the other transaction is made the deadlock victim
-- and is terminated with the error message discussed above.
Commit Transaction


/************************************************************************
*79****************** DEADLOCK VICTIM SELECTION ********************79*
************************************************************************/

-- 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, please watch Part 78 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



/************************************************************
*80****************** LOGGING DEADLOCKS ********************80*
************************************************************/

-- In this video we will discuss how to write the deadlock information to the SQL Server error log

-- When deadlocks occur, SQL Server chooses one of the transactions as the deadlock victim and rolls
it back.
-- There are several ways in SQL Server to track down the queries that are causing deadlocks.
-- One of the options is to use SQL Server trace flag 1222 to write the deadlock information to the SQL
-- Server error log.

-- Enable Trace flag : To enable trace flags use DBCC command. -1 parameter indicates that the trace
flag
-- must be set at the global level. If you omit -1 parameter the trace flag will be set only at the session
level.

DBCC Traceon(1222, -1)

--To check the status of the trace flag
DBCC TraceStatus(1222, -1)

-- To turn off the trace flag
DBCC Traceoff(1222, -1)

--The following SQL code generates a dead lock. This is the same code we discussed in Part 78 of SQL
Server Tutorial.

--SQL script to create the tables and populate them with test data
Create table TableA
(
```

```
    Id int identity primary key,
    Name nvarchar(50)
)
Go

Insert into TableA values ('Mark')
Go

Create table TableB
(
    Id int identity primary key,
    Name nvarchar(50)
)
Go

Insert into TableB values ('Mary')
Go

--SQL Script to create stored procedures
Create procedure spTransaction1
as
Begin
    Begin Tran
    Update TableA Set Name = 'Mark Transaction 1' where Id = 1
    Waitfor delay '00:00:05'
    Update TableB Set Name = 'Mary Transaction 1' where Id = 1
    Commit Transaction
End

Create procedure spTransaction2
as
Begin
    Begin Tran
    Update TableB Set Name = 'Mark Transaction 2' where Id = 1
    Waitfor delay '00:00:05'
    Update TableA Set Name = 'Mary Transaction 2' where Id = 1
    Commit Transaction
End

-- Open 2 instances of SQL Server Management studio. From the first window execute spTransaction1
-- and from the second window execute spTransaction2 back to back.

-- After a few seconds notice that one of the transactions complete successfully while the other
-- transaction is made the deadlock victim and rollback.

-- The information about this deadlock should now have been logged in sql server error log.

-- To read the error log
execute sp_readerrorlog
```

/**
*81********** DEADLOCK ANALYSIS AND PREVENTION
**********81*
**/

-- In this video we will discuss how to read and analyze sql server deadlock information captured in the error
-- log, so we can understand what's causing the deadlocks and take appropriate actions to prevent or minimize
-- the occurrence of deadlocks. This is continuation to Part 80. Please watch Part 80 from SQL Server
-- tutorial before proceeding.

-- The deadlock information in the error log has three sections
 Section Description
-- Deadlock Victim Contains the ID of the process that was selected as the deadlock victim and killed by SQL Server.
-- Process List Contains the list of the processes that participated in the deadlock.
-- Resource List Contains the list of the resources (database objects) owned by the processes involved in the deadlock

-- Process List : The process list has lot of items. Here are some of them that are particularly useful in understanding what caused the deadlock.

-- Node Description
-- loginname The loginname associated with the process
-- isolationlevel What isolation level is used
-- procname The stored procedure name
-- Inputbuf The code the process is executing when the deadlock occured

-- Resource List : Some of the items in the resource list that are particularly useful in understanding what caused the deadlock.

-- Node Description
-- objectname Fully qualified name of the resource involved in the deadlock
-- owner-list Contains (owner id) the id of the owning process and the lock mode it has acquired on the resource. lock mode determines how the resource can be accessed by concurrent transactions. S for Shared lock, U for Update lock, X for Exclusive lock etc
-- waiter-list Contains (waiter id) the id of the process that wants to acquire a lock on the resource and the lock mode it is requesting

To prevent the deadlock that we have in our case, we need to ensure that database objects (Table A & Table B) are accessed in the same order every time.
That is one of the processes will have to wait for the other to complete and not get stuck in a deadlock situation.

/**

*82********** CAPTURE DEADLOACK GRAPH USING SQL PROFILER**********82*
**********/

In this video we will discuss how to capture deadlock graph using SQL profiler.

To capture deadlock graph, all you need to do is add Deadlock graph event to the trace in SQL profiler.

Here are the steps :
1. Open SQL Profiler
2. Click File - New Trace. Provide the credentials and connect to the server
3. On the general tab, select "Blank" template from "Use the template" dropdownlist
sql profiler capture deadlocks

4. On the "Events Selection" tab, expand "Locks" section and select "Deadlock graph" event
sql profiler trace deadlock

5. Finally click the Run button to start the trace
6. At this point execute the code that causes deadlock
7. The deadlock graph should be captured in the profiler as shown below.
deadlock graph sql server profiler

The deadlock graph data is captured in XML format. If you want to extract this XML data to a physical file for later analysis,
you can do so by following the steps below.
1. In SQL profiler, click on "File - Export - Extract SQL Server Events - Extract Deadlock Events"
2. Provide a name for the file
3. The extension for the deadlock xml file is .xdl
4. Finally choose if you want to export all events in a single file or each event in a separate file

The deadlock information in the XML file is similar to what we have captured using the trace flag 1222.

Analyzing the deadlock graph
1. The oval on the graph, with the blue cross, represents the transaction that was chosen as the deadlock victim by SQL Server.
2. The oval on the graph represents the transaction that completed successfully.
3. When you move the mouse pointer over the oval, you can see the SQL code that was running that caused the deadlock.
4. The oval symbols represent the process nodes
Server Process Id : If you are using SQL Server Management Studio you can see the server process id on information bar at the bottom.
Deadlock Priority : If you have not set DEADLOCK PRIORITY explicitly using SET DEADLOCK PRIORITY statement, then both the processes
should have the same default deadlock priority NORMAL (0).
Log Used : The transaction log space used. If a transaction has used a lot of log space then the cost to roll it back is also more.
So the transaction that has used the least log space is killed and rolled back.
5. The rectangles represent the resource nodes.

HoBt ID : Heap Or Binary Tree ID. Using this ID query sys.partitions view to find the database objects involved in the deadlock.

```
SELECT object_name([object_id])
FROM sys.partitions
WHERE hobt_id = 72057594041663488
```

6. The arrows represent types of locks each process has on each resource node.

```
/*****************************************************************************
*83****************** SQL SERVER DEADLOCK ERROR HANDLING
********************83*
*****************************************************************************/
```

In this video we will discuss how to catch deadlock error using try/catch in SQL Server.
Modify the stored procedure as shown below to catch the deadlock error. The code is commented and is self-explanatory.

```
Alter procedure spTransaction1
as
Begin
   Begin Tran
   Begin Try
      Update TableA Set Name = 'Mark Transaction 1' where Id = 1
      Waitfor delay '00:00:05'
      Update TableB Set Name = 'Mary Transaction 1' where Id = 1
      -- If both the update statements succeeded.
      -- No Deadlock occurred. So commit the transaction.
      Commit Transaction
      Select 'Transaction Successful'
   End Try
   Begin Catch
      -- Check if the error is deadlock error
      If(ERROR_NUMBER() = 1205)
      Begin
         Select 'Deadlock. Transaction failed. Please retry'
      End
      -- Rollback the transaction
      Rollback
   End Catch
End

Alter procedure spTransaction2
as
Begin
   Begin Tran
   Begin Try
      Update TableB Set Name = 'Mary Transaction 2' where Id = 1
      Waitfor delay '00:00:05'
      Update TableA Set Name = 'Mark Transaction 2' where Id = 1
```

```
        Commit Transaction
        Select 'Transaction Successful'
    End Try
    Begin Catch
        If(ERROR_NUMBER() = 1205)
        Begin
            Select 'Deadlock. Transaction failed. Please retry'
        End
        Rollback
    End Catch
End
```

After modifying the stored procedures, execute both the procedures from 2 different windows simultaneously. Notice that the deadlock error is handled by the catch block.

In our next video, we will discuss how applications using ADO.NET can handle deadlock errors.

/**
*84********** HANDLING DEADLOCKS IN ADO.NET **********84*
**/

In this video we will discuss how to handle deadlock errors in an ADO.NET application.

In this video we will discuss how to capture deadlock graph using SQL profiler.

To handle deadlock errors in ADO.NET
1. Catch the SqlException object
2. Check if the error is deadlock error using the Number property of the SqlException object

Stored Procedure 1 Code

```
Alter procedure spTransaction1
as
Begin
    Begin Tran
    Update TableA Set Name = 'Mark Transaction 1' where Id = 1
    Waitfor delay '00:00:05'
    Update TableB Set Name = 'Mary Transaction 1' where Id = 1
    Commit Transaction
End
```

Stored Procedure 2 Code

```
Alter procedure spTransaction2
as
Begin
    Begin Tran
    Update TableB Set Name = 'Mark Transaction 2' where Id = 1
    Waitfor delay '00:00:05'
    Update TableA Set Name = 'Mary Transaction 2' where Id = 1
    Commit Transaction
```

End

WebForm1.aspx HTML

```
<table>
   <tr>
      <td>
         <asp:Button ID="Button1" runat="server"
            Text="Update Table A and then Table B"
            OnClick="Button1_Click" />
      </td>
   </tr>
   <tr>
      <td>
         <asp:Label ID="Label1" runat="server"></asp:Label>
      </td>
   </tr>
</table>
```

WebForm1.aspx.cs code

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace Demo
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      { }

      protected void Button1_Click(object sender, EventArgs e)
      {
         try
         {
            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
            using (SqlConnection con = new SqlConnection(cs))
            {
               SqlCommand cmd = new SqlCommand("spTransaction1", con);
               cmd.CommandType = CommandType.StoredProcedure;
               con.Open();
               cmd.ExecuteNonQuery();
               Label1.Text = "Transaction successful";
               Label1.ForeColor = System.Drawing.Color.Green;
            }
         }
         catch (SqlException ex)
         {
            if (ex.Number == 1205)
```

```
            {
                Label1.Text = "Deadlock. Please retry";
            }
            else
            {
                Label1.Text = ex.Message;
            }
            Label1.ForeColor = System.Drawing.Color.Red;
        }
    }
  }
}
```

WebForm2.aspx HTML

```
<table>
  <tr>
    <td>
      <asp:Button ID="Button1" runat="server"
        Text="Update Table B and then Table A"
        OnClick="Button1_Click" />
    </td>
  </tr>
  <tr>
    <td>
      <asp:Label ID="Label1" runat="server"></asp:Label>
    </td>
  </tr>
</table>
```

WebForm2.aspx.cs code

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace Demo
{
  public partial class WebForm1 : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    { }

    protected void Button1_Click(object sender, EventArgs e)
    {
      try
      {
        string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
        using (SqlConnection con = new SqlConnection(cs))
        {
```

```
                SqlCommand cmd = new SqlCommand("spTransaction1", con);
                cmd.CommandType = CommandType.StoredProcedure;
                con.Open();
                cmd.ExecuteNonQuery();
                Label1.Text = "Transaction successful";
                Label1.ForeColor = System.Drawing.Color.Green;
            }
        }
        catch (SqlException ex)
        {
            if (ex.Number == 1205)
            {
                Label1.Text = "Deadlock. Please retry";
            }
            else
            {
                Label1.Text = ex.Message;
            }
            Label1.ForeColor = System.Drawing.Color.Red;
        }
    }
  }
}
```

--85-- RETRY LOGIC FOR DEADLOCK EXCEPTIONS

In this video we will discuss implementing retry logic for deadlock exceptions.

This is continuation to Part 84. Please watch Part 84, before proceeding.

When a transaction fails due to deadlock, we can write some logic so the system can resubmit the transaction. The deadlocks usually last for a very short duration. So upon resubmitting the transaction it may complete successfully. This is much better from user experience standpoint.

To achieve this we will be using the following technologies
C#
ASP.NET
SQL Server
jQuery AJAX

Result.cs

```
public class Result
{
   public int AttemptsLeft { get; set; }
   public string Message { get; set; }
   public bool Success { get; set; }
}
```

WebForm1.aspx HTML and jQuery code

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <script src="jquery-1.11.2.js"></script>
    <script type="text/javascript">
        $(document).ready(function () {
            var lblMessage = $('#Label1');
            var attemptsLeft;

            function updateData() {
                $.ajax({
                    url: 'WebForm1.aspx/CallStoredProcedure',
                    method: 'post',
                    contentType: 'application/json',
                    data: '{attemptsLeft:' + attemptsLeft + '}',
                    dataType: 'json',
                    success: function (data) {
                        lblMessage.text(data.d.Message);
                        attemptsLeft = data.d.AttemptsLeft;
                        if (data.d.Success) {
                            $('#btn').prop('disabled', false);
                            lblMessage.css('color','green');
                        }
                        else if(attemptsLeft > 0){
                            lblMessage.css('color', 'red');
                            updateData();
                        }
                        else {
                            lblMessage.css('color', 'red');
                            lblMessage.text('Deadlock Occurred. ZERO attempts left. Please try later');
                        }
                    },
                    error: function (err) {
                        lblMessage.css('color', 'red');
                        lblMessage.text(err.responseText);
                    }
                });
            }

            $('#btn').click(function () {
                $(this).prop('disabled', true);
                lblMessage.text('Updating....');
                attemptsLeft = 5;
                updateData();
            });
        });
```

```
    </script>
</head>
<body style="font-family: Arial">
    <form id="form1" runat="server">
        <input id="btn" type="button"
            value="Update Table A and then Table B" />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>
```

WebForm1.aspx.cs code

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace Demo
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        { }

        [System.Web.Services.WebMethod]
        public static Result CallStoredProcedure(int attemptsLeft)
        {
            Result _result = new Result();
            if (attemptsLeft > 0)
            {
                try
                {
                    string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
                    using (SqlConnection con = new SqlConnection(cs))
                    {
                        SqlCommand cmd = new SqlCommand("spTransaction15", con);
                        cmd.CommandType = CommandType.StoredProcedure;
                        con.Open();
                        cmd.ExecuteNonQuery();
                        _result.Message = "Transaction successful";
                        _result.AttemptsLeft = 0;
                        _result.Success = true;
                    }
                }
                catch (SqlException ex)
                {
                    if (ex.Number == 1205)
                    {
```

```
                _result.AttemptsLeft = attemptsLeft - 1;
                _result.Message = "Deadlock occurred. Retrying. Attempts left : "
                    + _result.AttemptsLeft.ToString();
            }
            else
            {
                throw;
            }
            _result.Success = false;
        }
    }
    return _result;
  }
 }
}
```

Copy and paste the above code in WebForm2.aspx and make the required changes as described in the video.

--86-- HOW TO FIND BLOCKING QUERIES IN SQL SERVER

In this video we will discuss, how to find blocking queries in sql server.

Blocking occurs if there are open transactions. Let us understand this with an example.

Execute the following 2 sql statements
Begin Tran
Update TableA set Name='Mark Transaction 1' where Id = 1

Now from a different window, execute any of the following commands. Notice that all the queries are blocked.
Select Count(*) from TableA
Delete from TableA where Id = 1
Truncate table TableA
Drop table TableA

This is because there is an open transaction. Once the open transaction completes, you will be able to execute the above queries.

So the obvious next question is - How to identify all the active transactions.

One way to do this is by using DBCC OpenTran. DBCC OpenTran will display only the oldest active transaction. It is not going to show you all the open transactions.
DBCC OpenTran

The following link has the SQL script that you can use to identify all the active transactions.
http://www.sqlskills.com/blogs/paul/script-open-transactions-with-text-and-plans

The beauty about this script is that it has a lot more useful information about the open transactions

Session Id
Login Name
Database Name
Transaction Begin Time
The actual query that is executed

You can now use this information and ask the respective developer to either commit or rollback the transactions that they have left open unintentionally.

For some reason if the person who initiated the transaction is not available, you also have the option to KILL the associated process. However, this may have unintended consequences, so use it with extreme caution.

There are 2 ways to kill the process are described below

Killing the process using SQL Server Activity Monitor :
1. Right Click on the Server Name in Object explorer and select "Activity Monitor"
2. In the "Activity Monitor" window expand Processes section
3. Right click on the associated "Session ID" and select "Kill Process" from the context menu

Killing the process using SQL command :
KILL Process_ID

What happens when you kill a session
All the work that the transaction has done will be rolled back. The database must be put back in the state it was in, before the transaction started.

--87-- SQL SERVER EXCEPT OPERATOR

In this video we will discuss SQL Server except operator with examples.

EXCEPT operator returns unique rows from the left query that aren't in the right query's results.
Introduced in SQL Server 2005
The number and the order of the columns must be the same in all queries
The data types must be same or compatible
This is similar to minus operator in oracle
Let us understand this with an example. We will use the following 2 tables for this example.

SQL Script to create the tables

```
Create Table TableA
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go
```

```
Insert into TableA values (1, 'Mark', 'Male')
Insert into TableA values (2, 'Mary', 'Female')
Insert into TableA values (3, 'Steve', 'Male')
Insert into TableA values (4, 'John', 'Male')
Insert into TableA values (5, 'Sara', 'Female')
Go

Create Table TableB
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go

Insert into TableB values (4, 'John', 'Male')
Insert into TableB values (5, 'Sara', 'Female')
Insert into TableB values (6, 'Pam', 'Female')
Insert into TableB values (7, 'Rebeka', 'Female')
Insert into TableB values (8, 'Jordan', 'Male')
Go
```

Notice that the following query returns the unique rows from the left query that aren't in the right query's results.

```
Select Id, Name, Gender
From TableA
Except
Select Id, Name, Gender
From TableB
```

Result :
sql server except operator result

To retrieve all of the rows from Table B that does not exist in Table A, reverse the two queries as shown below.

```
Select Id, Name, Gender
From TableB
Except
Select Id, Name, Gender
From TableA
```

Result :
reverse result

You can also use Except operator on a single table. Lets use the following tblEmployees table for this example.
Employees table

SQL script to create tblEmployees table

```
Create table tblEmployees
(
   Id int identity primary key,
   Name nvarchar(100),
   Gender nvarchar(10),
   Salary int
)
Go

Insert into tblEmployees values ('Mark', 'Male', 52000)
Insert into tblEmployees values ('Mary', 'Female', 55000)
Insert into tblEmployees values ('Steve', 'Male', 45000)
Insert into tblEmployees values ('John', 'Male', 40000)
Insert into tblEmployees values ('Sara', 'Female', 48000)
Insert into tblEmployees values ('Pam', 'Female', 60000)
Insert into tblEmployees values ('Tom', 'Male', 58000)
Insert into tblEmployees values ('George', 'Male', 65000)
Insert into tblEmployees values ('Tina', 'Female', 67000)
Insert into tblEmployees values ('Ben', 'Male', 80000)
Go
```

Result :
single table except result

Order By clause should be used only once after the right query

```
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 50000
Except
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 60000
order By Name
```

--88-- DIFFERENCE BETWEEN EXCEPT AND NOT IN SQL SERVER

In this video we will discuss the difference between EXCEPT and NOT IN operators in SQL Server.

We will use the following 2 tables for this example.

The following query returns the rows from the left query that aren’t in the right query’s results.

```
Select Id, Name, Gender From TableA
Except
Select Id, Name, Gender From TableB
```

Result :
except operator result

The same result can also be achieved using NOT IN operator.
Select Id, Name, Gender From TableA
Where Id NOT IN (Select Id from TableB)

So, what is the difference between EXCEPT and NOT IN operators
1. Except filters duplicates and returns only DISTINCT rows from the left query that aren't in the right query's results, where as NOT IN does not filter the duplicates.

Insert the following row into TableA
Insert into TableA values (1, 'Mark', 'Male')

Now execute the following EXCEPT query. Notice that we get only the DISTINCT rows
Select Id, Name, Gender From TableA
Except
Select Id, Name, Gender From TableB

Result:
except operator result

Now execute the following query. Notice that the duplicate rows are not filtered.
Select Id, Name, Gender From TableA
Where Id NOT IN (Select Id from TableB)

Result:
sql server not in example

2. EXCEPT operator expects the same number of columns in both the queries, where as NOT IN, compares a single column from the outer query with a single column from the subquery.

In the following example, the number of columns are different.
Select Id, Name, Gender From TableA
Except
Select Id, Name From TableB

The above query would produce the following error.
Msg 205, Level 16, State 1, Line 1
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

NOT IN, compares a single column from the outer query with a single column from subquery.

In the following example, the subquery returns multiple columns
Select Id, Name, Gender From TableA
Where Id NOT IN (Select Id, Name from TableB)

Msg 116, Level 16, State 1, Line 2

Only one expression can be specified in the select list when the subquery is not introduced with EXISTS.

--89-- INTERSECT OPERATOR IN SQL SERVER

In this video we will discuss
1. Intersect operator in sql server
2. Difference between intersect and inner join

Intersect operator retrieves the common records from both the left and the right query of the Intersect operator.
Introduced in SQL Server 2005
The number and the order of the columns must be same in both the queries
The data types must be same or at least compatible
Let us understand INTERSECT operator with an example.

We will use the following 2 tables for this example.

SQL Script to create the tables and populate with test data

```
Create Table TableA
(
   Id int,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go

Insert into TableA values (1, 'Mark', 'Male')
Insert into TableA values (2, 'Mary', 'Female')
Insert into TableA values (3, 'Steve', 'Male')
Go

Create Table TableB
(
   Id int,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go

Insert into TableB values (2, 'Mary', 'Female')
Insert into TableB values (3, 'Steve', 'Male')
Go
```

The following query retrieves the common records from both the left and the right query of the Intersect operator.

```
Select Id, Name, Gender from TableA
Intersect
Select Id, Name, Gender from TableB
```

Result :
intersect operator in sql example

We can also achieve the same thinkg using INNER join. The following INNER join query would produce the exact same result.

```
Select TableA.Id, TableA.Name, TableA.Gender
From TableA Inner Join TableB
On TableA.Id = TableB.Id
```

What is the difference between INTERSECT and INNER JOIN
1. INTERSECT filters duplicates and returns only DISTINCT rows that are common between the LEFT
and Right Query, where as INNER JOIN does not filter the duplicates.

To understand this difference, insert the following row into TableA

```
Insert into TableA values (2, 'Mary', 'Female')
```

Now execute the following INTERSECT query. Notice that we get only the DISTINCT rows

```
Select Id, Name, Gender from TableA
Intersect
Select Id, Name, Gender from TableB
```

Result :
intersect operator in sql example

Now execute the following INNER JOIN query. Notice that the duplicate rows are not filtered.

```
Select TableA.Id, TableA.Name, TableA.Gender
From TableA Inner Join TableB
On TableA.Id = TableB.Id
```

Result :
inner join sql duplicate rows

You can make the INNER JOIN behave like INTERSECT operator by using the DISTINCT operator

```
Select DISTINCT TableA.Id, TableA.Name, TableA.Gender
From TableA Inner Join TableB
On TableA.Id = TableB.Id
```

Result :
inner join remove duplicate rows

2. INNER JOIN treats two NULLS as two different values. So if you are joining two tables based on a nullable column and if both tables have NULLs in that joining column then, INNER JOIN will not include those rows in the result-set, where as INTERSECT treats two NULLs as a same value and it returns all matching rows.

To understand this difference, execute the following 2 insert statements
Insert into TableA values(NULL, 'Pam', 'Female')
Insert into TableB values(NULL, 'Pam', 'Female')

INTERSECT query
Select Id, Name, Gender from TableA
Intersect
Select Id, Name, Gender from TableB

Result :
sql intersect null values

INNER JOIN query
Select TableA.Id, TableA.Name, TableA.Gender
From TableA Inner Join TableB
On TableA.Id = TableB.Id

--90-- DIFFERENCE BETWEEN UNION INTERESECT AND EXCEPT IN SQL SERVER

In this video we will discuss the difference between union intersect and except in sql server with examples.

UNION operator returns all the unique rows from both the left and the right query. UNION ALL included the duplicates as well.

INTERSECT operator retrieves the common unique rows from both the left and the right query.

EXCEPT operator returns unique rows from the left query that aren’t in the right query’s results.

Let us understand these differences with examples. We will use the following 2 tables for the examples.

SQL Script to create the tables
```
Create Table TableA
(
   Id int,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go
```

```
Insert into TableA values (1, 'Mark', 'Male')
Insert into TableA values (2, 'Mary', 'Female')
Insert into TableA values (3, 'Steve', 'Male')
Insert into TableA values (3, 'Steve', 'Male')
Go

Create Table TableB
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go

Insert into TableB values (2, 'Mary', 'Female')
Insert into TableB values (3, 'Steve', 'Male')
Insert into TableB values (4, 'John', 'Male')
Go
```

UNION operator returns all the unique rows from both the queries. Notice the duplicates are removed.

```
Select Id, Name, Gender from TableA
UNION
Select Id, Name, Gender from TableB
```

Result :
sql server union example

UNION ALL operator returns all the rows from both the queries, including the duplicates.

```
Select Id, Name, Gender from TableA
UNION ALL
Select Id, Name, Gender from TableB
```

Result :
sql server union all example

INTERSECT operator retrieves the common unique rows from both the left and the right query. Notice the duplicates are removed.

```
Select Id, Name, Gender from TableA
INTERSECT
Select Id, Name, Gender from TableB
```

Result :
sql server intersect example

EXCEPT operator returns unique rows from the left query that aren’t in the right query’s results.

```
Select Id, Name, Gender from TableA
EXCEPT
Select Id, Name, Gender from TableB
```

Result :
sql server except example

If you wnat the rows that are present in Table B but not in Table A, reverse the queries.

```
Select Id, Name, Gender from TableB
EXCEPT
Select Id, Name, Gender from TableA
```

Result :
except operator in sql server

For all these 3 operators to work the following 2 conditions must be met
The number and the order of the columns must be same in both the queries
The data types must be same or at least compatible
For example, if the number of columns are different, you will get the following error
Msg 205, Level 16, State 1, Line 1
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

--91-- CROSS APPLY AND OUTER APPLY IN SQL SERVER

In this video we will discuss cross apply and outer apply in sql server with examples.

We will use the following 2 tables for examples in this demo
department table employee table

SQL Script to create the tables and populate with test data

```
Create table Department
(
   Id int primary key,
   DepartmentName nvarchar(50)
)
Go

Insert into Department values (1, 'IT')
Insert into Department values (2, 'HR')
Insert into Department values (3, 'Payroll')
Insert into Department values (4, 'Administration')
Insert into Department values (5, 'Sales')
Go

Create table Employee
```

```
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10),
   Salary int,
   DepartmentId int foreign key references Department(Id)
)
Go

Insert into Employee values (1, 'Mark', 'Male', 50000, 1)
Insert into Employee values (2, 'Mary', 'Female', 60000, 3)
Insert into Employee values (3, 'Steve', 'Male', 45000, 2)
Insert into Employee values (4, 'John', 'Male', 56000, 1)
Insert into Employee values (5, 'Sara', 'Female', 39000, 2)
Go
```

We want to retrieve all the matching rows between Department and Employee tables.
sql server inner join example

This can be very easily achieved using an Inner Join as shown below.

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Inner Join Employee E
On D.Id = E.DepartmentId
```

Now if we want to retrieve all the matching rows between Department and Employee tables + the non-matching rows from the LEFT table (Department)
sql server left join example

This can be very easily achieved using a Left Join as shown below.

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Left Join Employee E
On D.Id = E.DepartmentId
```

Now lets assume we do not have access to the Employee table. Instead we have access to the following Table Valued function, that returns
all employees belonging to a department by Department Id.

```
Create function fn_GetEmployeesByDepartmentId(@DepartmentId int)
Returns Table
as
Return
(
   Select Id, Name, Gender, Salary, DepartmentId
   from Employee where DepartmentId = @DepartmentId
)
Go
```

The following query returns the employees of the department with Id =1.
Select * from fn_GetEmployeesByDepartmentId(1)

Now if you try to perform an Inner or Left join between Department table and
fn_GetEmployeesByDepartmentId()
function you will get an error.

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Inner Join fn_GetEmployeesByDepartmentId(D.Id) E
On D.Id = E.DepartmentId
```

If you execute the above query you will get the following error
Msg 4104, Level 16, State 1, Line 3
The multi-part identifier "D.Id" could not be bound.

This is where we use Cross Apply and Outer Apply operators. Cross Apply is semantically equivalent to
Inner Join and Outer Apply is semantically equivalent to Left Outer Join.

Just like Inner Join, Cross Apply retrieves only the matching rows from the Department table and fn_GetEmployeesByDepartmentId() table valued function.

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Cross Apply fn_GetEmployeesByDepartmentId(D.Id) E
```

Just like Left Outer Join, Outer Apply retrieves all matching rows from the Department table and fn_GetEmployeesByDepartmentId() table valued function + non-matching rows from the left table (Department)

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Outer Apply fn_GetEmployeesByDepartmentId(D.Id) E
```

How does Cross Apply and Outer Apply work
The APPLY operator introduced in SQL Server 2005, is used to join a table to a table-valued function. The Table Valued Function on the right hand side of the APPLY operator gets called for each row from the left (also called outer table) table.
Cross Apply returns only matching rows (semantically equivalent to Inner Join)
Outer Apply returns matching + non-matching rows (semantically equivalent to Left Outer Join). The unmatched columns of the table valued function will be set to NULL.

--92-- DDL TRIGGERS IN SQL SERVER

In this video we will discuss DDL Triggers in sql server.

In SQL Server there are 4 types of triggers
1. DML Triggers - Data Manipulation Language. Discussed in Parts 43 to 47 of SQL Server Tutorial.
2. DDL Triggers - Data Definition Language
3. CLR triggers - Common Language Runtime
4. Logon triggers

What are DDL triggers
DDL triggers fire in response to DDL events - CREATE, ALTER, and DROP (Table, Function, Index, Stored Procedure etc...).
For the list of all DDL events please visit https://msdn.microsoft.com/en-us/library/bb522542.aspx

Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.
Example - sp_rename system stored procedure

What is the use of DDL triggers
If you want to execute some code in response to a specific DDL event
To prevent certain changes to your database schema
Audit the changes that the users are making to the database structure
Syntax for creating DDL trigger

```
CREATE TRIGGER [Trigger_Name]
ON [Scope (Server|Database)]
FOR [EventType1, EventType2, EventType3, ...],
AS
BEGIN
  -- Trigger Body
END
```

DDL triggers scope : DDL triggers can be created in a specific database or at the server level.

The following trigger will fire in response to CREATE_TABLE DDL event.

```
CREATE TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE
AS
BEGIN
  Print 'New table created'
END
```

To check if the trigger has been created
In the Object Explorer window, expand the SampleDB database by clicking on the plus symbol.
Expand Programmability folder
Expand Database Triggers folder
find ddl triggers sql server

Please note : If you cant find the trigger that you just created, make sure to refresh the Database Triggers folder.

When you execute the following code to create the table, the trigger will automatically fire and will print the message -
New table created
Create Table Test (Id int)

The above trigger will be fired only for one DDL event CREATE_TABLE. If you want this trigger to be fired for multiple events,
for example when you alter or drop a table, then separate the events using a comma as shown below.

```
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
  Print 'A table has just been created, modified or deleted'
END
```

Now if you create, alter or drop a table, the trigger will fire automatically and you will get the message -
A table has just been created, modified or deleted.

The 2 DDL triggers above execute some code in response to DDL events

Now let us look at an example of how to prevent users from creating, altering or dropping tables.
To do this modify the trigger as shown below.

```
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
  Rollback
  Print 'You cannot create, alter or drop a table'
END
```

To be able to create, alter or drop a table, you either have to disable or delete the trigger.

To disable trigger
1. Right click on the trigger in object explorer and select "Disable" from the context menu
2. You can also disable the trigger using the following T-SQL command

```
DISABLE TRIGGER trMyFirstTrigger ON DATABASE
```

To enable trigger
1. Right click on the trigger in object explorer and select "Enable" from the context menu
2. You can also enable the trigger using the following T-SQL command

```
ENABLE TRIGGER trMyFirstTrigger ON DATABASE
```

To drop trigger

1. Right click on the trigger in object explorer and select "Delete" from the context menu
2. You can also drop the trigger using the following T-SQL command

```
DROP TRIGGER trMyFirstTrigger ON DATABASE
```

Certain system stored procedures that perform DDL-like operations can also fire DDL triggers. The following trigger will be fired when ever you rename a database object using sp_rename system stored procedure.

```
CREATE TRIGGER trRenameTable
ON DATABASE
FOR RENAME
AS
BEGIN
   Print 'You just renamed something'
END
```

The following code changes the name of the TestTable to NewTestTable. When this code is executed, it will fire the trigger trRenameTable

```
sp_rename 'TestTable', 'NewTestTable'
```

The following code changes the name of the Id column in NewTestTable to NewId. When this code is executed, it will fire the trigger trRenameTable

```
sp_rename 'NewTestTable.Id' , 'NewId', 'column'
```

Server-scoped ddl triggers
Suggested Videos
Part 90 - Difference between union intersect and except in sql server
Part 91 - Cross apply and outer apply in sql server
Part 92 - DDL Triggers in sql server

--93-- SERVER SCOPED DDL TRIGGERS

In this video we will discuss server-scoped ddl triggers

The following trigger is a database scoped trigger. This will prevent users from creating, altering or dropping tables only from the database in which it is created.

```
CREATE TRIGGER tr_DatabaseScopeTrigger
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   ROLLBACK
   Print 'You cannot create, alter or drop a table in the current database'
END
```

If you have another database on the server, they will be able to create, alter or drop tables in

that database. If you want to prevent users from doing this you may create the trigger again in this database.

But, what if you have 100 different databases on your SQL Server, and you want to prevent users from creating,
altering or dropping tables from all these 100 databases. Creating the same trigger for all the 100 different
databases is not a good approach for 2 reasons.
1. It is tedious and error prone
2. Maintainability is a night mare. If for some reason you have to change the trigger, you will have to do it
in 100 different databases, which again is tedious and error prone.

This is where server-scoped DDL triggers come in handy. When you create a server scoped DDL trigger,
it will fire in response to the DDL events happening in all of the databases on that server.

Creating a Server-scoped DDL trigger : Similar to creating a database scoped trigger, except that you will have
to change the scope to ALL Server as shown below.

```
CREATE TRIGGER tr_ServerScopeTrigger
ON ALL SERVER
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   ROLLBACK
   Print 'You cannot create, alter or drop a table in any database on the server'
END
```

Now if you try to create, alter or drop a table in any of the databases on the server, the trigger will be fired.

Where can I find the Server-scoped DDL triggers
1. In the Object Explorer window, expand "Server Objects" folder
2. Expand Triggers folder
Server-scoped ddl triggers

To disable Server-scoped ddl trigger
1. Right click on the trigger in object explorer and select "Disable" from the context menu
2. You can also disable the trigger using the following T-SQL command

```
DISABLE TRIGGER tr_ServerScopeTrigger ON ALL SERVER
```

To enable Server-scoped ddl trigger
1. Right click on the trigger in object explorer and select "Enable" from the context menu
2. You can also enable the trigger using the following T-SQL command

```
ENABLE TRIGGER tr_ServerScopeTrigger ON ALL SERVER
```

To drop Server-scoped ddl trigger

1. Right click on the trigger in object explorer and select "Delete" from the context menu
2. You can also drop the trigger using the following T-SQL command

```
DROP TRIGGER tr_ServerScopeTrigger ON ALL SERVER
```

--94-- SQL SERVER TRIGGER EXECUTION ORDER

In this video we will discuss how to set the execution order of triggers using sp_settriggerorder stored procedure.

Server scoped triggers will always fire before any of the database scoped triggers. This execution order cannot be changed.

In the example below, we have a database-scoped and a server-scoped trigger handling the same event (CREATE_TABLE).
When you create a table, notice that server-scoped trigger is always fired before the database-scoped trigger.

```
CREATE TRIGGER tr_DatabaseScopeTrigger
ON DATABASE
FOR CREATE_TABLE
AS
BEGIN
   Print 'Database Scope Trigger'
END
GO

CREATE TRIGGER tr_ServerScopeTrigger
ON ALL SERVER
FOR CREATE_TABLE
AS
BEGIN
   Print 'Server Scope Trigger'
END
GO
```

Using the sp_settriggerorder stored procedure, you can set the execution order of server-scoped or database-scoped triggers.

sp_settriggerorder stored procedure has 4 parameters

Parameter	Description
@triggername	Name of the trigger
@order	Value can be First, Last or None. When set to None, trigger is fired in random order
@stmttype	SQL statement that fires the trigger. Can be INSERT, UPDATE, DELETE or any DDL event
@namespace	Scope of the trigger. Value can be DATABASE, SERVER, or NULL

```
EXEC sp_settriggerorder
@triggername = 'tr_DatabaseScopeTrigger1',
@order = 'none',
@stmttype = 'CREATE_TABLE',
@namespace = 'DATABASE'
GO
```

If you have a database-scoped and a server-scoped trigger handling the same event, and if you have set the execution order at
both the levels. Here is the execution order of the triggers.
1. The server-scope trigger marked First
2. Other server-scope triggers
3. The server-scope trigger marked Last
4. The database-scope trigger marked First
5. Other database-scope triggers
6. The database-scope trigger marked Last

--95-- AUDIT TABLE CHANGES IN SQL SERVER

In this video we will discuss, how to audit table changes in SQL Server using a DDL trigger.

Table to store the audit data

```
Create table TableChanges
(
   DatabaseName nvarchar(250),
   TableName nvarchar(250),
   EventType nvarchar(250),
   LoginName nvarchar(250),
   SQLCommand nvarchar(2500),
   AuditDateTime datetime
)
Go
```

The following trigger audits all table changes in all databases on a SQL Server

```
CREATE TRIGGER tr_AuditTableChanges
ON ALL SERVER
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   DECLARE @EventData XML
   SELECT @EventData = EVENTDATA()

   INSERT INTO SampleDB.dbo.TableChanges
   (DatabaseName, TableName, EventType, LoginName,
    SQLCommand, AuditDateTime)
```

```
    VALUES
    (
        @EventData.value('(/EVENT_INSTANCE/DatabaseName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/LoginName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2500)'),
        GetDate()
    )
END
```

In the above example we are using EventData() function which returns event data in XML format. The following XML is returned by the EventData() function when I created a table with name = MyTable in SampleDB database.

```
<EVENT_INSTANCE>
  <EventType>CREATE_TABLE</EventType>
  <PostTime>2015-09-11T16:12:49.417</PostTime>
  <SPID>58</SPID>
  <ServerName>VENKAT-PC</ServerName>
  <LoginName>VENKAT-PC\Tan</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>SampleDB</DatabaseName>
  <SchemaName>dbo</SchemaName>
  <ObjectName>MyTable</ObjectName>
  <ObjectType>TABLE</ObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON"
                ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON"
                ENCRYPTED="FALSE" />
    <CommandText>
      Create Table MyTable
      (
        Id int,
        Name nvarchar(50),
        Gender nvarchar(50)
      )
    </CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```