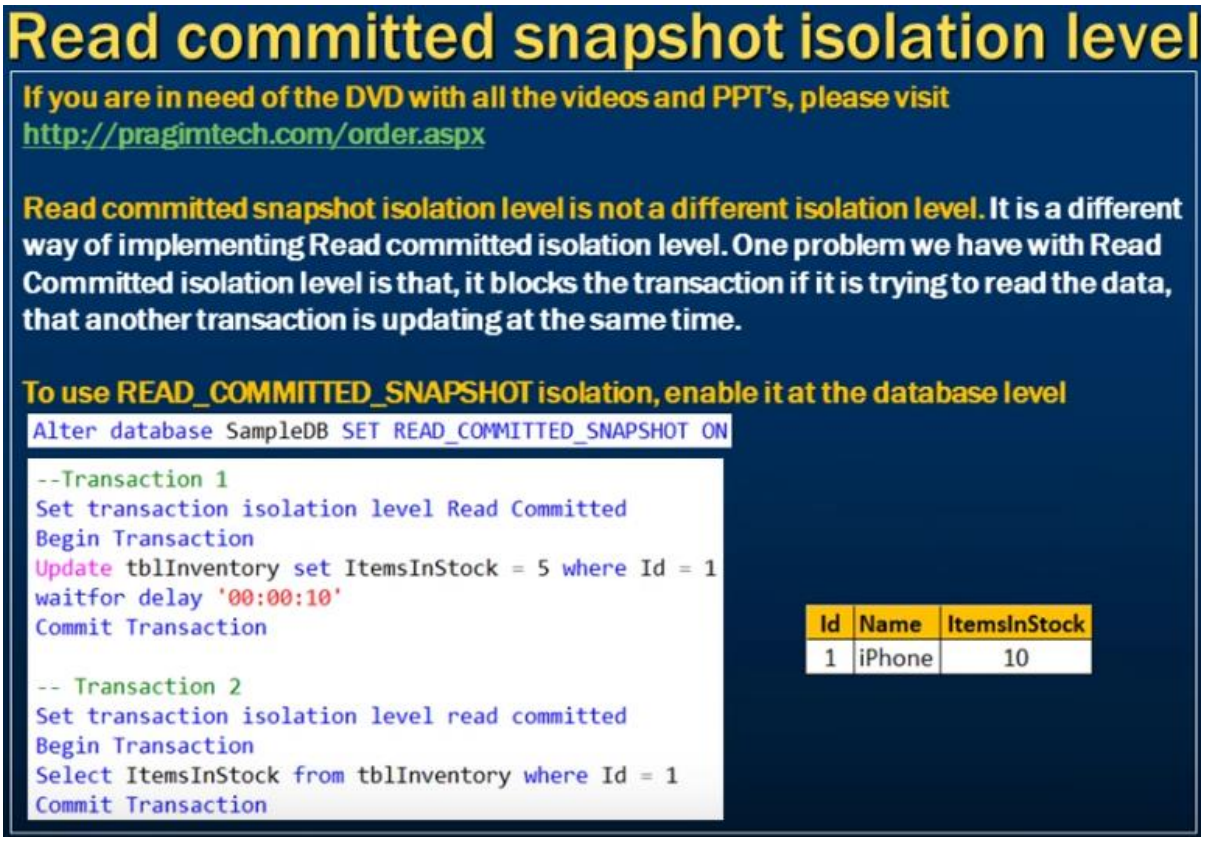
Read committed snapshot isolation level

READ COMMITTED SNAPSHOT TRANASACTION ISOLATION LEVEL WORKS SIMILAR TO SNAPSHOT TRANSACTION ISOLATION LEVEL.



Below if we execute the update part of Transaction 1 but not the Commit part and then execute Transaction 2 then Transaction 2 will be blocked because it is set to read committed and it is trying to read the same data that Transaction 1 is updating.

After Commit Transaction of Transaction 1 is executed, Transaction 2 will also get executed.

```
Commit Transaction
```

To change the behavior of Transaction 2 we enable Read_committed_snapshot_on:
Note that to alter this connection 1 has to be first closed, so we close instance 1 SSMS.

```
ALter database SampleDB SET READ_COMMITTED_SNAPSHOT O

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

READ COMMITTED SNAPSHOT ISOLATION LEVEL ALSO ALLOWS UPDATES TO HAPPEN BY A SECOND TRANSACTION BUT IT IS KEPT ON HOLD UNTIL THE 1ST TRANSACTION COMPLETES AND ONLY TAKES PLACE AFTER THE FIRST TRANSACTION HAS COMPLETED. THEREFORE, WE DO NOT HAVE UPDATE CONFLICTS WITH READ COMMITTED SNAPSHOT ISOLATION LEVEL.

```
Alter database SampleDB
SET ALLOW_SNAPSHOT_ISOLATION OFF

Alter database SampleDB
SET READ_COMMITTED_SNAPSHOT ON

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
    Update tblInventory
    set ItemsInStock = 8 where Id = 1

Commit Transaction

Select * from tblInventory
```

Messages
Command(s) completed successfully.

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

Messages
(1 row(s) affected)

READ COMMITTED SNAPSHOT ISOLATION LEVEL ALLOWS US TO READ THE DATA THAT HAS BEEN LAST COMMITTED AND NOT THE ONLY THAT IS STORED AS COPY IN TEMP DB. THEREFORE THIS IS ONE BIG DIFFERENCE BETWEEN SNAPSHOT TRANSACTION ISOLATION LEVEL AND READ COMMITTED TRANSACTION ISOLATION LEVEL.

READ COMMITTED SNAPSHOT:

```
Alter database SampleDB
SET ALLOW_SNAPSHOT_ISOLATION OFF

Alter database SampleDB
SET READ_COMMITTED_SNAPSHOT ON

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
    Update tblInventory
    set ItemsInStock = 8 where Id = 1

Commit Transaction

Select * from tblInventory
```

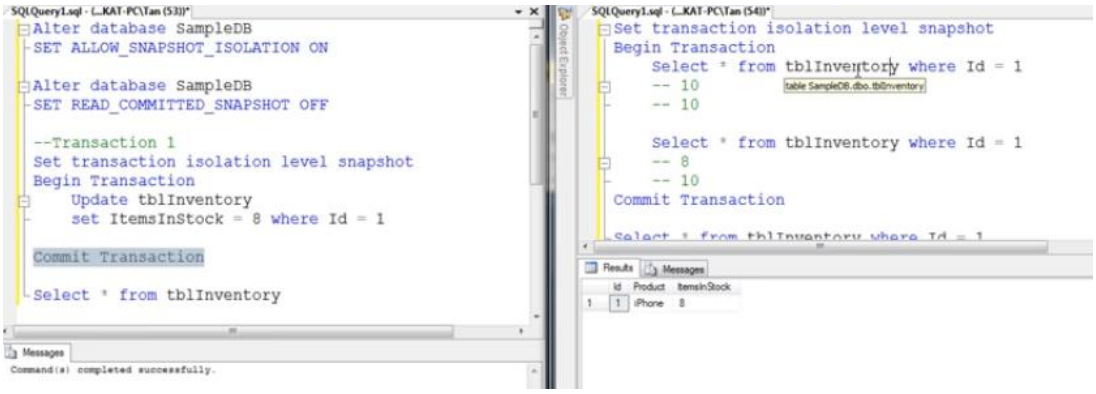
Messages
Command(s) completed successfully.

```
-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
    Select * from tblInventory where Id = 1
    -- 10

    Select * from tblInventory where Id = 1
    -- 8
Commit Transaction
```

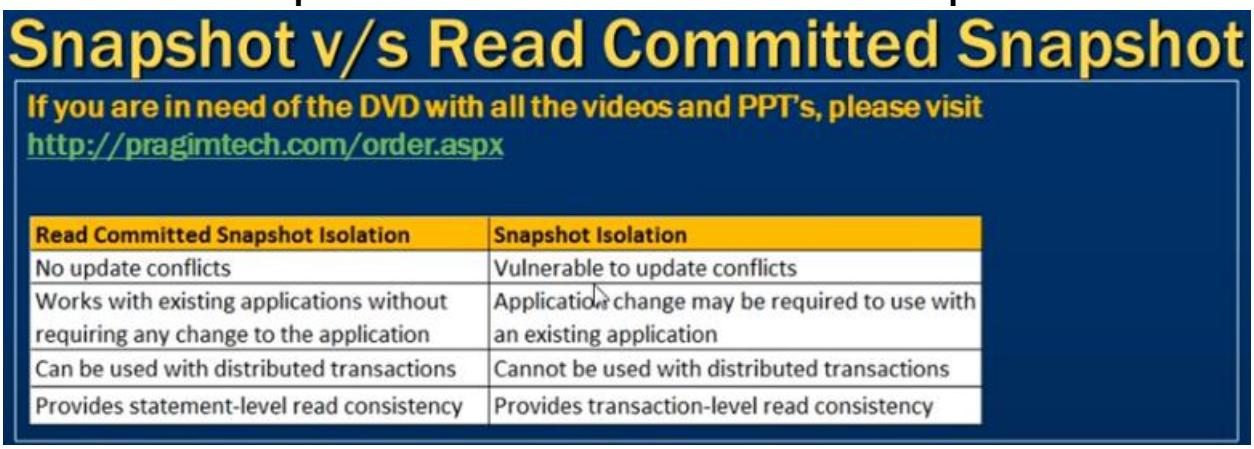
Results | Messages

	Id	Product	ItemsInStock
1	1	iPhone	8

SNAPSHOT ISOLATION LEVEL:



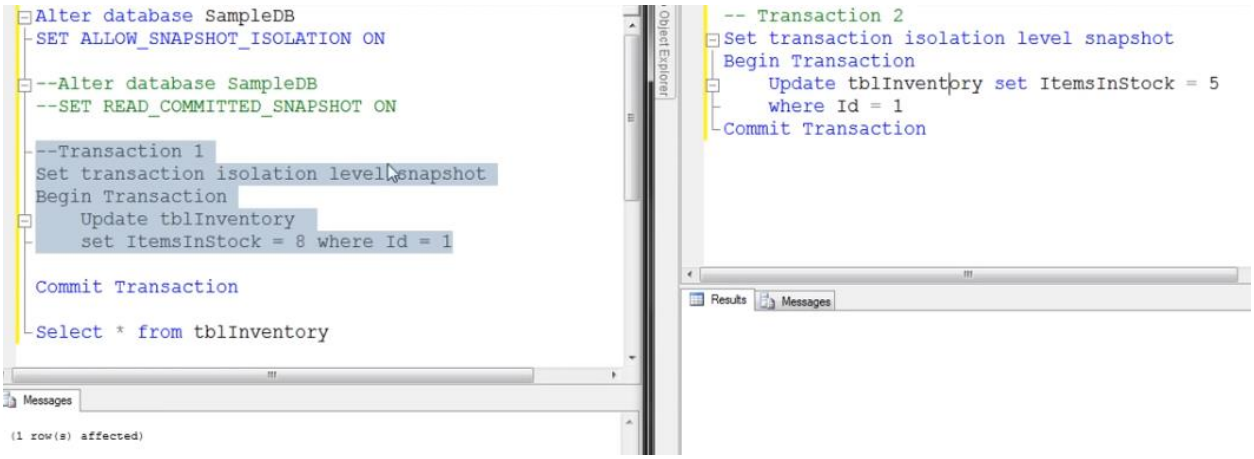
Difference between snapshot isolation and read committed snapshot



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

Snapshot Isolation is vulnerable to update conflict whereas Read Committed Snapshot Isolation is not. If a snapshot isolation is trying to update a piece of data that another transaction is updating at the same time, then an update conflict occurs, the transaction terminates and rollbacks with an error.

We have two instances of SSMS here. If we do not commit transaction 1 then transaction 2 will be blocked as it is trying to update the same data.



After we commit transaction 1 then transaction 2 also completes but with error.

```
Alter database SampleDB
SET ALLOW_SNAPSHOT_ISOLATION ON

--Alter database SampleDB
--SET READ_COMMITTED_SNAPSHOT ON

--Transaction 1
Set transaction isolation level snapshot
Begin Transaction
    Update tblInventory
    set ItemsInStock = 8 where Id = 1

Commit Transaction

Select * from tblInventory
```

Messages
Command(s) completed successfully.

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

Messages
Msg 3960, Level 16, State 2, Line 4
Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot

Now, let's turn snapshot_isolation off and turn on read_committed_snapshot. Make the changes shown below and execute the transactions 1 excluding commit part and transaction 2.

```
Alter database SampleDB
SET ALLOW_SNAPSHOT_ISOLATION OFF

Alter database SampleDB
SET READ_COMMITTED_SNAPSHOT ON

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
    Update tblInventory
    set ItemsInStock = 8 where Id = 1

Commit Transaction

Select * from tblInventory
```

Messages
(1 row(s) affected)

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

Results | Messages

Next we run Commit Transaction in 1.

```
Commit Transaction
```

This causes Transaction 2 also to become unblocked and completes. Transaction 1 set the ItemsInStock to 8 and Transaction 2 set it to 5. And because Transaction 2 completed last hence it is 5.

```
Alter database SampleDB
SET ALLOW_SNAPSHOT_ISOLATION OFF

Alter database SampleDB
SET READ_COMMITTED_SNAPSHOT ON

--Transaction 1
Set transaction isolation level read committed
Begin Transaction
    Update tblInventory
    set ItemsInStock = 8 where Id = 1

Commit Transaction

Select * from tblInventory
```

Results | Messages

Id	Product	ItemsInStock
1	iPhone	5

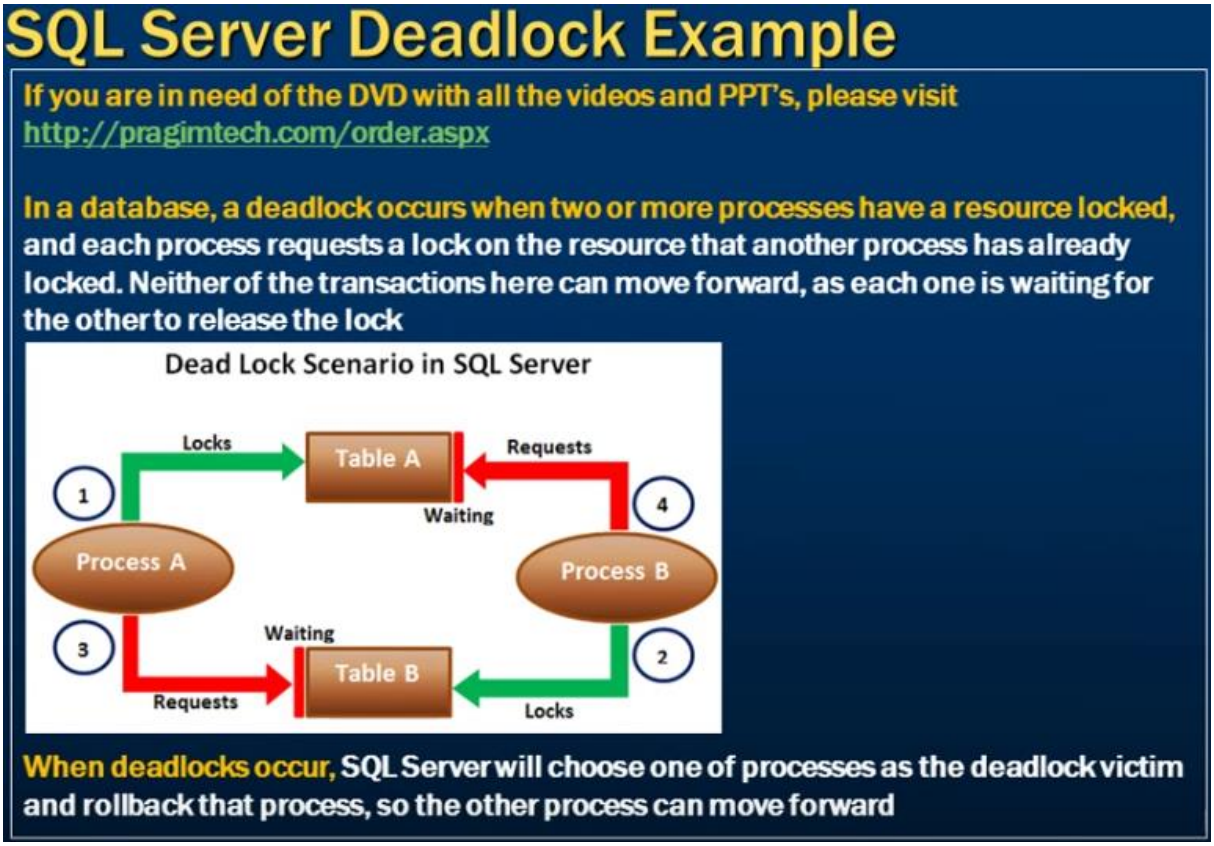
```
-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
    Update tblInventory set ItemsInStock = 5
    where Id = 1
Commit Transaction
```

Messages

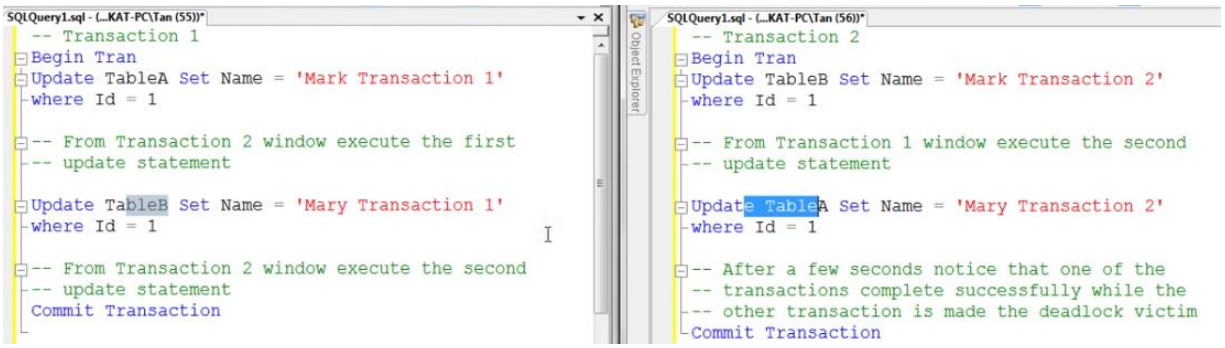
```
(1 row(s) affected)
```

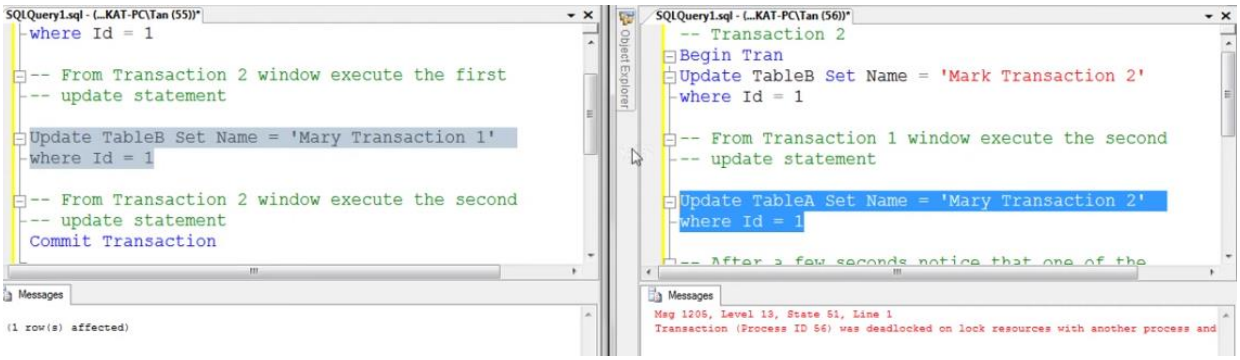
Thus with “Read committed Snapshot isolation” we don’t have update conflicts, but with we do have update conflicts with Snapshot isolation.

Deadlocks



BELOW TRANSACTION 2 GETS CHOSEN AS THE DEADLOCK VICTIM AND THE ENTIRE PROCESS GETS ROLLED BACK AND ONLY TRANSACTION 1 PROCEEDS.





WE GET THE FOLLOWING ERROR:

```
Msg 1205, Level 13, State 51, Line 1
Transaction (Process ID 56) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.
```

TRANSACTION 2 GETS CHOSEN AS THE DEADLOCK VICTIM BECAUSE IT IS EASIER TO ROLL BACK UPDATE IN 1 ROW IN TRANSACTION 2 RATHER THAN ROLLING BACK MULTIPLE ROWS IN TRANSACTION 1:

THERFORE, TRANSACTION THAT IS LEAST EXPENSIVE TO ROLL BACK OR HAS THE LOWEST DEADLOCK PRIORITY GETS CHOSEN AS THE DEADLOCK VICTIM:

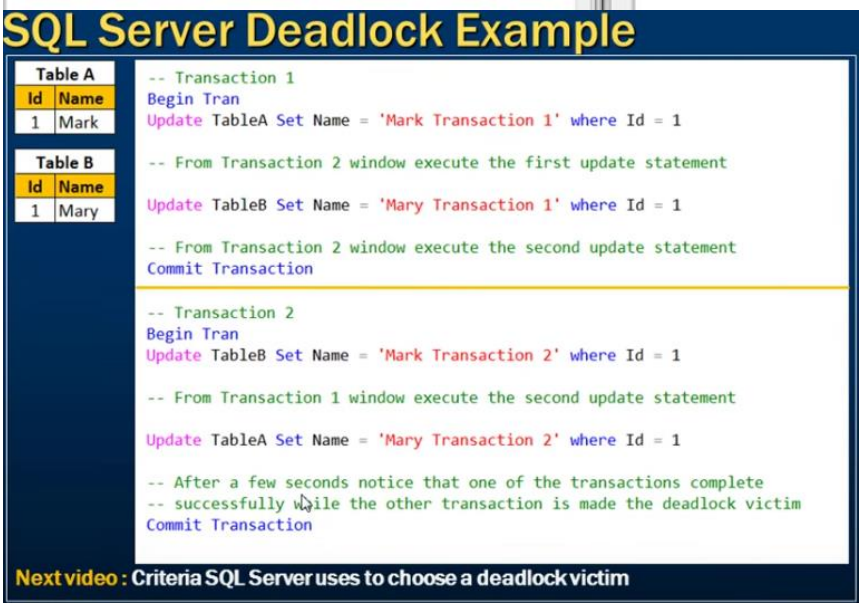
```
-- Transaction 1
Begin Tran

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

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

Commit Transaction
```

```
-- Transaction 2
Begin Tran

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

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

Commit Transaction
```

BUT WE CAN EXPLICITY PROVIDE DEADLOCK PRIORITY: BELOW WE CHOOSE TRANSACTION 2 AS THE DEADLOCK PRIORITY TO BE HIGH. THEREFORE, TRANSACTION 1 GET CHOSEN AS DEADLOCK VICTIM WHICH GETS LOWEST DEADLOCK PRIORITY

```
-- Transaction 1
Begin Tran

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

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

Commit Transaction
```

```
-- Transaction 2
SET DEADLOCK_PRIORITY HIGH
Begin Tran

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

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

Commit Transaction
```



Deadlock victim selection

- How SQL Server detects deadlocks
- What happens when a deadlock is detected
- What is DEADLOCK_PRIORITY
- What is the criteria that SQL Server uses to choose a deadlock victim when there is a deadlock

SQL Server deadlock victim selection

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

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

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

Deadlock Victim Selection Criteria

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

Below Transaction 1 wants to update TableA first then TableB, and Transaction 2 wants to update TableB first then TableA. We have not set any deadlock priority.

```
-- Transaction 1
Begin Tran

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

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

Commit Transaction
```

```
-- Transaction 2
Begin Tran

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

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

Commit Transaction
```

First we run Transaction 1 and update TableA, it works, tableA updated.

```
-- Transaction 1
Begin Tran

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

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

Commit Transaction
```

```
-- Transaction 2
Begin Tran

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

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

Commit Transaction
```

Messages

(5 row(s) affected)

Next we execute Transaction 2 and update TableB, it works, tableB updated.

```
-- Transaction 1
Begin Tran

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

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

Commit Transaction
```

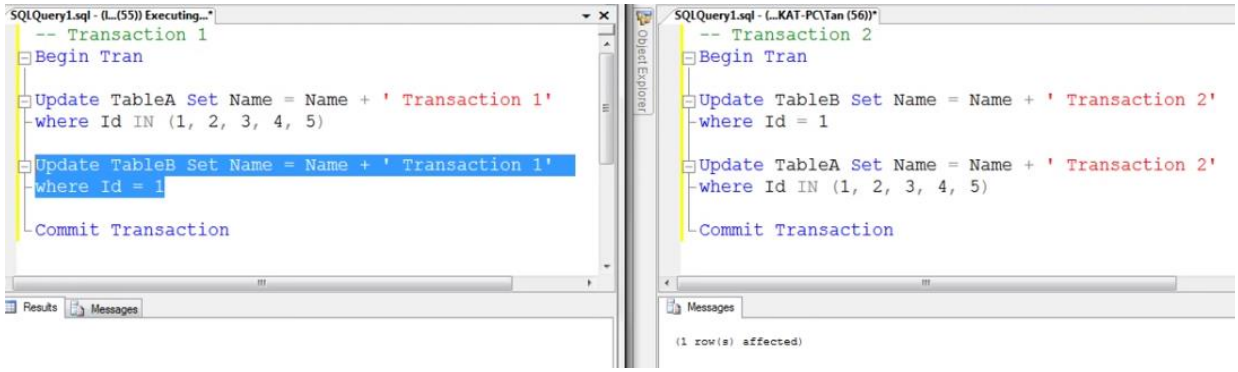
```
-- Transaction 2
Begin Tran

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

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

Commit Transaction
```

Messages

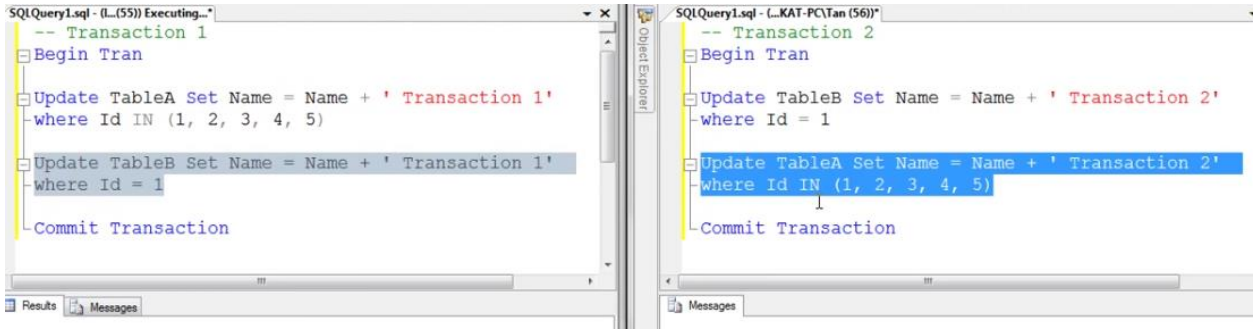
(5 row(s) affected)

(1 row(s) affected)

Next when we try to execute Transaction 1 and update TableB, it gets blocked. This is because Transaction 2 has already acquired lock on TableB.

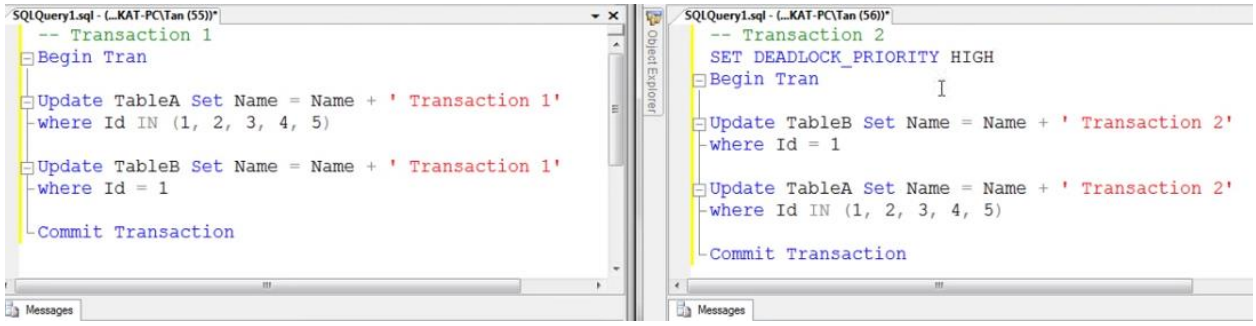


Now if we try to execute Transaction 2 and update TableA, there will be deadlock situation



Now which Transaction will be chosen as deadlock victim, it will be Transaction 2 because it is the one least expensive to roll back because it has updated only one row while Transaction updated TableA with 5 rows.

However, we can change the behavior by setting deadlock priority and Transaction 1 will be chosen as victim.



Deadlock Victim Selection - Example 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
```

Logging deadlocks

- How to write the deadlock information to the SQL Server error log

By default deadlock information is not written to error log. When deadlock occurs SQL Server chooses one of the transactions as the victim and rolls it back.

Logging deadlocks in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

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.

```
-- Set SQL Server trace flag 1222
DBCC Traceon(1222, -1)

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

-- Turn off the trace flag
DBCC Traceoff(1222, -1)
```

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

Please note:-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

Turn on at session level:

```
DBCC Traceon(1222)

DBCC TraceStatus(1222)

DBCC Traceoff(1222)
```

Check the status, we find it turned on at session level:

```
DBCC Traceon(1222)

DBCC TraceStatus(1222)

DBCC Traceoff(1222)
```

	TraceFlag	Status	Global	Session
1	1222	1	0	1

To turn it on at global level we have to supply second parameter.

```
DBCC Traceon(1222, -1)

DBCC TraceStatus(1222)

DBCC Traceoff(1222)
```

Now if we check the status, we find it on for **global level**.

```
DBCC Traceon(1222, -1)

DBCC TraceStatus(1222)

DBCC Traceoff(1222)
```

	TraceFlag	Status	Global	Session
1	1222	1	1	0

To test the scenario, let us create two tables and two stored procedures, both the stored procedures will try to update both the tables.

```
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
```

```
Execute spTransaction1

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
```

```
Execute spTransaction2

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
```

Now we EXECUTE THE system stored procedure to read the ERROR LOG TO CHECK WHERE THE DEADLOCK TOOK PLACE:

```
execute sp_readerrorlog
```

	LogDate	ProcessInfo	Text
1	2015-08-27 19:06:45.310	spid54	Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (X64...
2	2015-08-27 19:06:45.310	spid54	(c) 2005 Microsoft Corporation.
3	2015-08-27 19:06:45.310	spid54	All rights reserved.
4	2015-08-27 19:06:45.310	spid54	Server process ID is 1452.
5	2015-08-27 19:06:45.310	spid54	System Manufacturer: 'Acer', System Model: 'Aspire 53...
6	2015-08-27 19:06:45.310	spid54	Authentication mode is MIXED.
7	2015-08-27 19:06:45.310	spid54	Logging SQL Server messages in file 'C:\Program Files...
8	2015-08-27 19:06:45.310	spid54	The error log has been reinitialized. See the previous lo...
9	2015-08-27 19:09:37.460	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...
10	2015-08-27 19:09:50.590	spid59	DBCC TRACEOFF 1222, server process ID (SPID) 59. ...
11	2015-08-27 19:10:04.680	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...
12	2015-08-27 19:10:54.650	spid59	DBCC TRACEOFF 1222, server process ID (SPID) 59. ...
13	2015-08-27 19:13:02.070	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...
14	2015-08-27 19:13:13.430	spid59	DBCC TRACEOFF 1222, server process ID (SPID) 59. ...
15	2015-08-27 19:13:18.350	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...

Deadlock analysis and prevention

- How to read and analyze sql server deadlock information captured in the error log

Deadlock analysis and prevention

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

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 Section :

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

When we run the system stored procedure we can read the error log, and if we enabled deadlock logging then its log will be there.

SQLQuery6.sql - (...KAT-PC\Tan (55))*

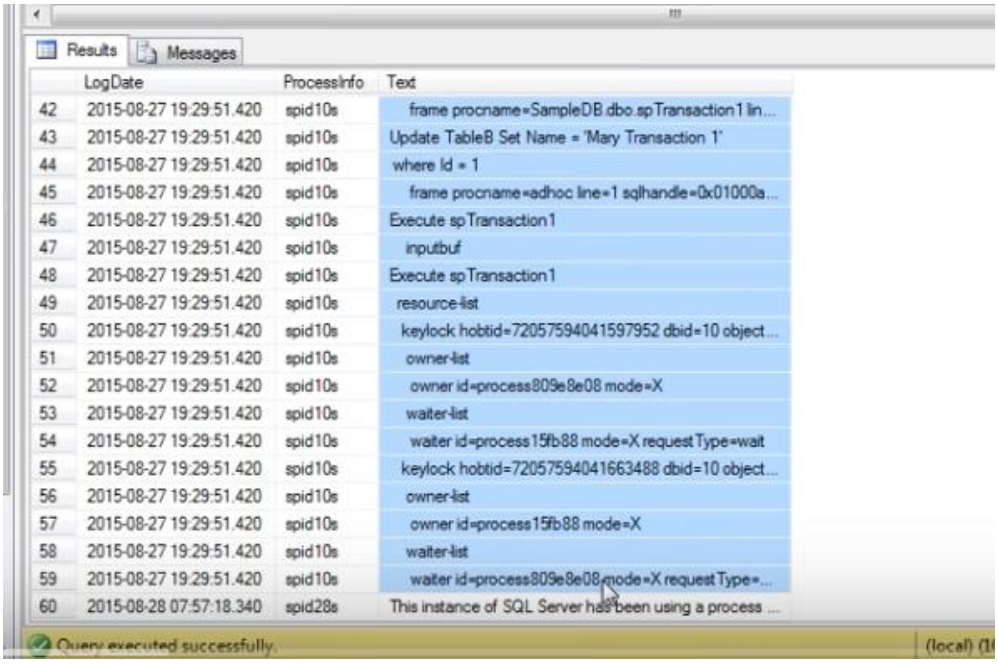
```
execute sp_readerrorlog
```

Results | Messages

	LogDate	ProcessInfo	Text
25	2015-08-27 19:28:06.490	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...
26	2015-08-27 19:28:19.330	spid59	DBCC TRACEOFF 1222, server process ID (SPID) 59. ...
27	2015-08-27 19:28:25.410	spid59	DBCC TRACEON 1222, server process ID (SPID) 59. ...
28	2015-08-27 19:29:51.420	spid10s	deadlock-list
29	2015-08-27 19:29:51.420	spid10s	deadlock victim=process15fb88
30	2015-08-27 19:29:51.420	spid10s	process-list
31	2015-08-27 19:29:51.420	spid10s	process id=process15fb88 taskpriority=0 logused=26...
32	2015-08-27 19:29:51.420	spid10s	executionStack
33	2015-08-27 19:29:51.420	spid10s	frame procname=SampleDB.dbo.spTransaction2 lin...
34	2015-08-27 19:29:51.420	spid10s	Update TableA Set Name = 'Mary Transaction 2'
35	2015-08-27 19:29:51.420	spid10s	where Id = 1
36	2015-08-27 19:29:51.420	spid10s	frame procname=adhoc line=1 sqlhandle=0x01000a...
37	2015-08-27 19:29:51.420	spid10s	Execute spTransaction2
38	2015-08-27 19:29:51.420	spid10s	inputbuf
39	2015-08-27 19:29:51.420	spid10s	Execute spTransaction2
40	2015-08-27 19:29:51.420	spid10s	process id=process809e8e08 taskpriority=0 logused=...
41	2015-08-27 19:29:51.420	spid10s	executionStack
42	2015-08-27 19:29:51.420	spid10s	frame procname=SampleDB.dbo.spTransaction1 lin...
43	2015-08-27 19:29:51.420	spid10s	Update TableB Set Name = 'Mary Transaction 1'

Query executed successfully. (local)

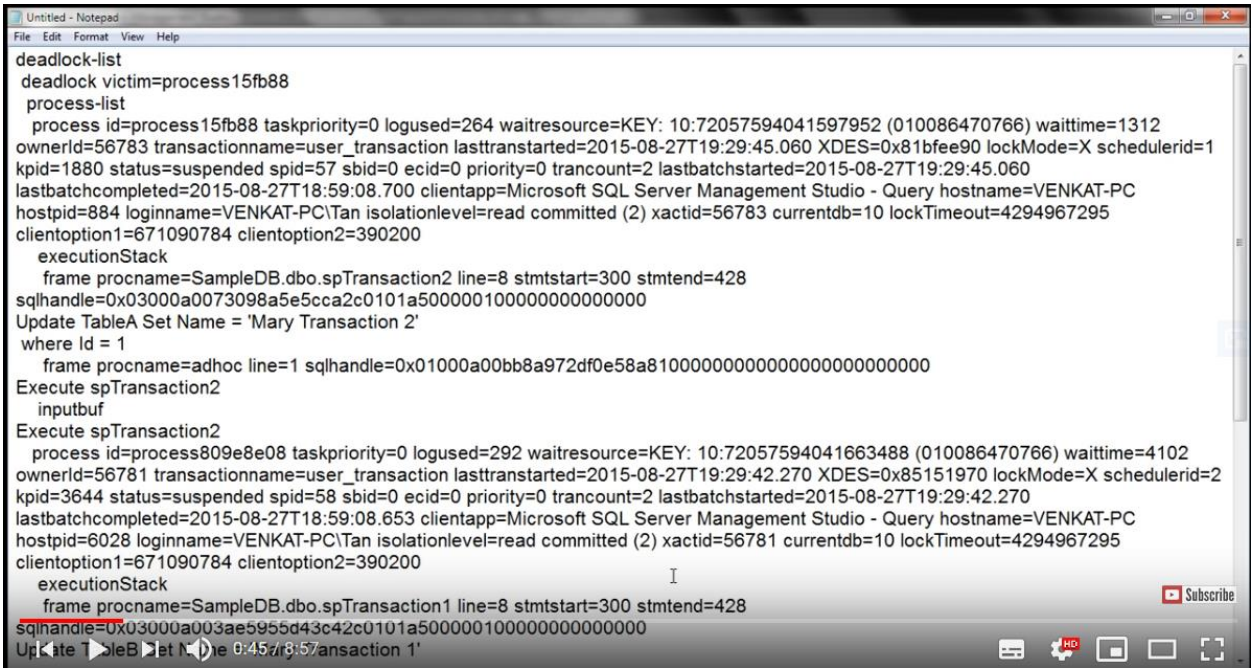
Let us copy these rows and paste them in notepad so that we can understand what is causing this deadlock.



Results | Messages

	LogDate	ProcessInfo	Text
42	2015-08-27 19:29:51.420	spid10s	frame procname=SampleDB.dbo.spTransaction1 lin...
43	2015-08-27 19:29:51.420	spid10s	Update TableB Set Name = 'Mary Transaction 1'
44	2015-08-27 19:29:51.420	spid10s	where Id = 1
45	2015-08-27 19:29:51.420	spid10s	frame procname=adhoc line=1 sqlhandle=0x01000a...
46	2015-08-27 19:29:51.420	spid10s	Execute spTransaction1
47	2015-08-27 19:29:51.420	spid10s	inputbuf
48	2015-08-27 19:29:51.420	spid10s	Execute spTransaction1
49	2015-08-27 19:29:51.420	spid10s	resource-list
50	2015-08-27 19:29:51.420	spid10s	keylock hobtid=72057594041597952 dbid=10 object...
51	2015-08-27 19:29:51.420	spid10s	owner-list
52	2015-08-27 19:29:51.420	spid10s	owner id=process809e8e08 mode=X
53	2015-08-27 19:29:51.420	spid10s	waiter-list
54	2015-08-27 19:29:51.420	spid10s	waiter id=process15fb88 mode=X requestType=wait
55	2015-08-27 19:29:51.420	spid10s	keylock hobtid=72057594041663488 dbid=10 object...
56	2015-08-27 19:29:51.420	spid10s	owner-list
57	2015-08-27 19:29:51.420	spid10s	owner id=process15fb88 mode=X
58	2015-08-27 19:29:51.420	spid10s	waiter-list
59	2015-08-27 19:29:51.420	spid10s	waiter id=process809e8e08 mode=X requestType=...
60	2015-08-28 07:57:18.340	spid28s	This instance of SQL Server has been using a process ...

Query executed successfully. (local) (1



Untitled - Notepad

File Edit Format View Help

```
deadlock-list
 deadlock victim=process15fb88
  process-list
   process id=process15fb88 taskpriority=0 logused=264 waitresource=KEY: 10:72057594041597952 (010086470766) waittime=1312
ownerId=56783 transactionname=user_transaction lasttranstarted=2015-08-27T19:29:45.060 XDES=0x81bfee90 lockMode=X schedulerid=1
kpid=1880 status=suspended spid=57 sbid=0 ecid=0 priority=0 trancount=2 lastbatchstarted=2015-08-27T19:29:45.060
lastbatchcompleted=2015-08-27T18:59:08.700 clientapp=Microsoft SQL Server Management Studio - Query hostname=VENKAT-PC
hostpid=884 loginname=VENKAT-PC\Tan isolationlevel=read committed (2) xactid=56783 currentdb=10 lockTimeout=4294967295
clientoption1=671090784 clientoption2=390200
    executionStack
     frame procname=SampleDB.dbo.spTransaction2 line=8 stmtstart=300 stmtend=428
sqlhandle=0x03000a0073098a5e5cca2c0101a50000010000000000000000
Update TableA Set Name = 'Mary Transaction 2'
 where Id = 1
     frame procname=adhoc line=1 sqlhandle=0x01000a00bb8a972df0e58a810000000000000000000000000
Execute spTransaction2
    inputbuf
Execute spTransaction2
   process id=process809e8e08 taskpriority=0 logused=292 waitresource=KEY: 10:72057594041663488 (010086470766) waittime=4102
ownerId=56781 transactionname=user_transaction lasttranstarted=2015-08-27T19:29:42.270 XDES=0x85151970 lockMode=X schedulerid=2
kpid=3644 status=suspended spid=58 sbid=0 ecid=0 priority=0 trancount=2 lastbatchstarted=2015-08-27T19:29:42.270
lastbatchcompleted=2015-08-27T18:59:08.653 clientapp=Microsoft SQL Server Management Studio - Query hostname=VENKAT-PC
hostpid=6028 loginname=VENKAT-PC\Tan isolationlevel=read committed (2) xactid=56781 currentdb=10 lockTimeout=4294967295
clientoption1=671090784 clientoption2=390200
    executionStack
     frame procname=SampleDB.dbo.spTransaction1 line=8 stmtstart=300 stmtend=428
sqlhandle=0x03000a003ae5955d43c42c0101a50000010000000000000000
Update TableB Set Name = 'Mary Transaction 1'
```

Subscribe

0:45 / 8:57

The first thing that we have to understand is that the deadlock information in the log has 3 sections

1. **Deadlock Victim**
2. **Process List**
3. **Resource List**

Deadlock analysis and prevention

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

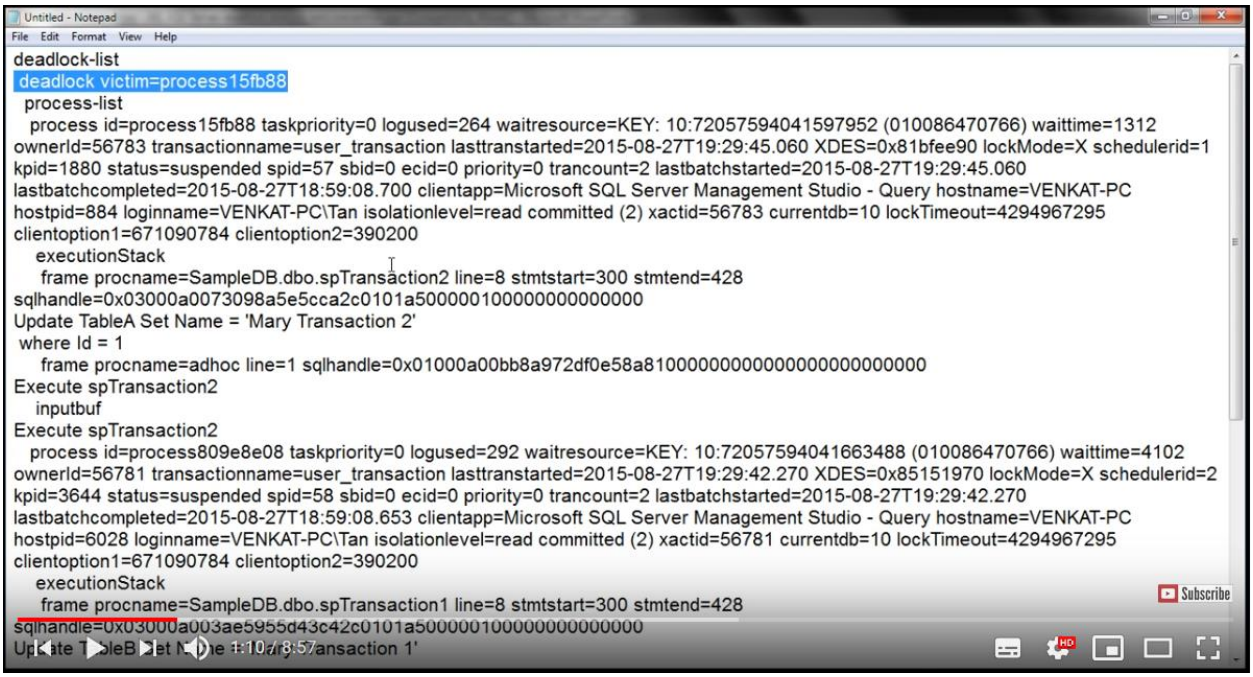
The deadlock information in the error log has three sections

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 Section :

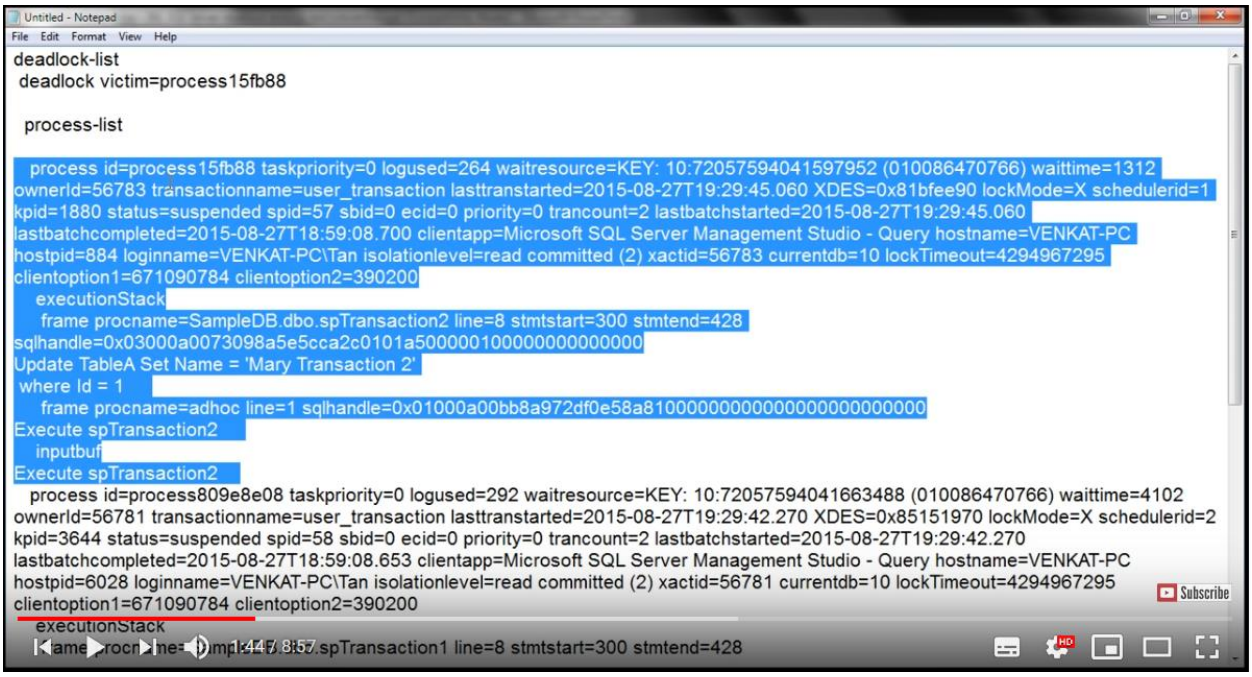
Node	Description
loginname	The loginname associated with the process
isolationlevel	What isolation level is used
procname	The stored procedure name
Inputbuf	The code the process is executing when the deadlock occured

The first thing that we observe is deadlock victim section, this contains ID of the process that is deadlock victim.

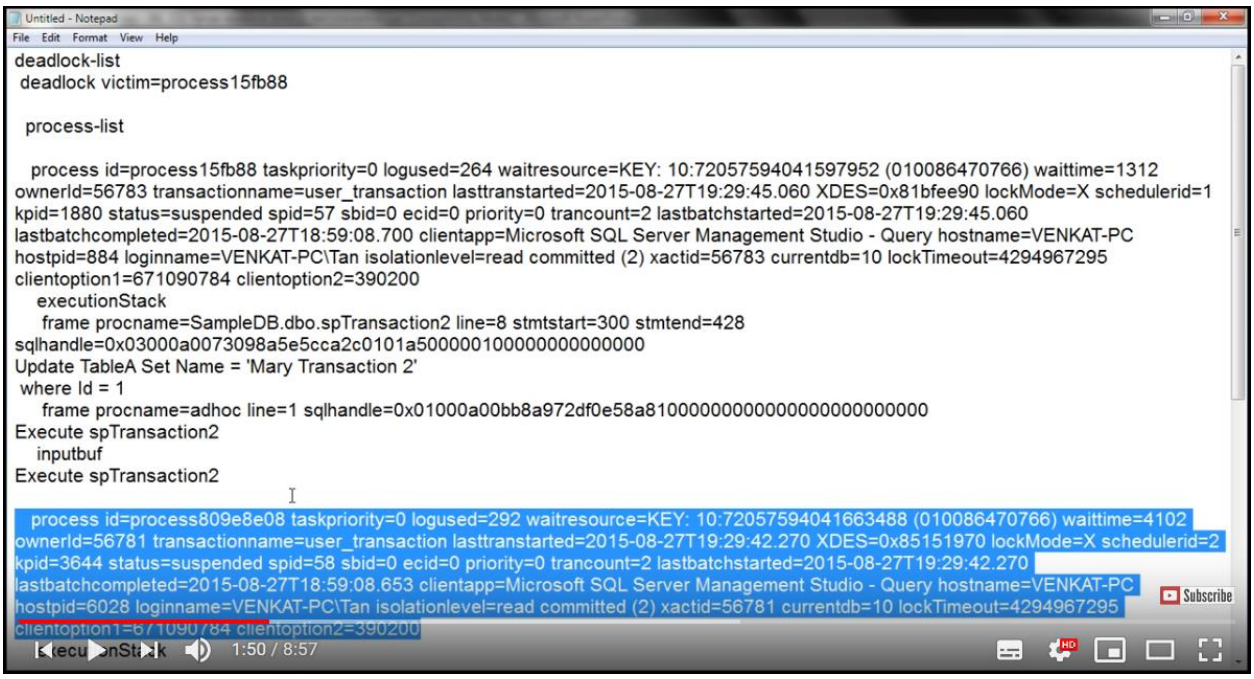


We take up this ID and look into the Process List section.

We can see the start of this Process List section and its end as below:



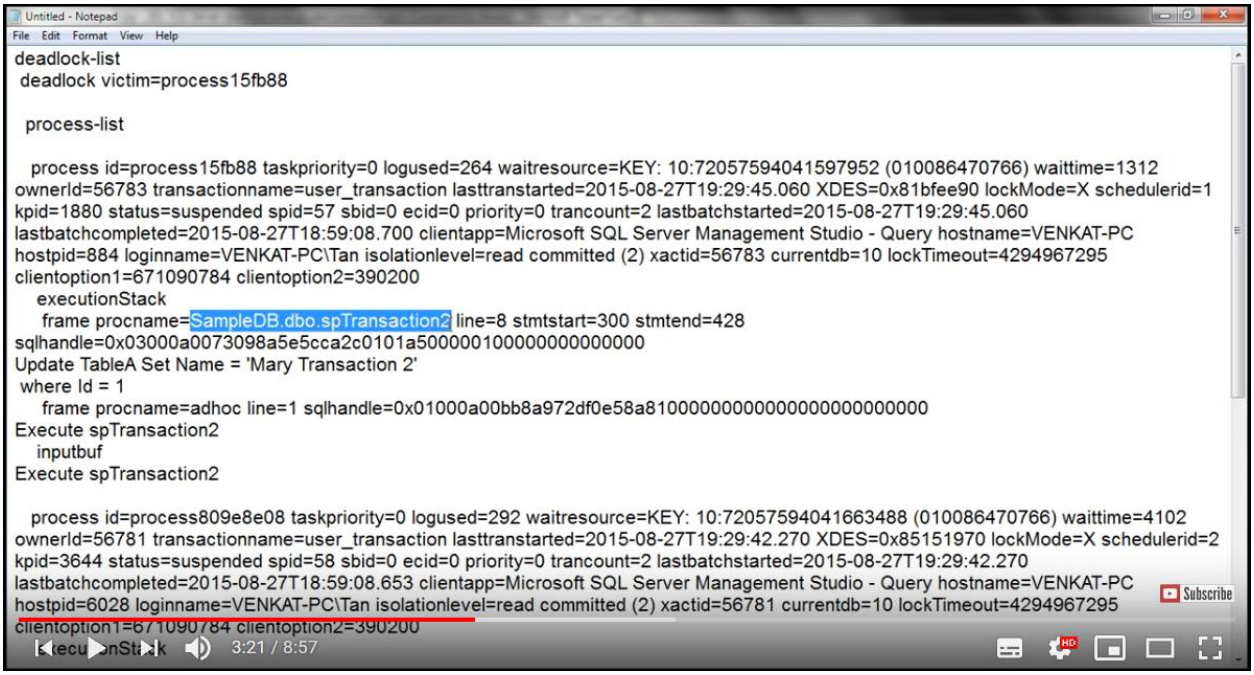
And the second process is listed right next to it.



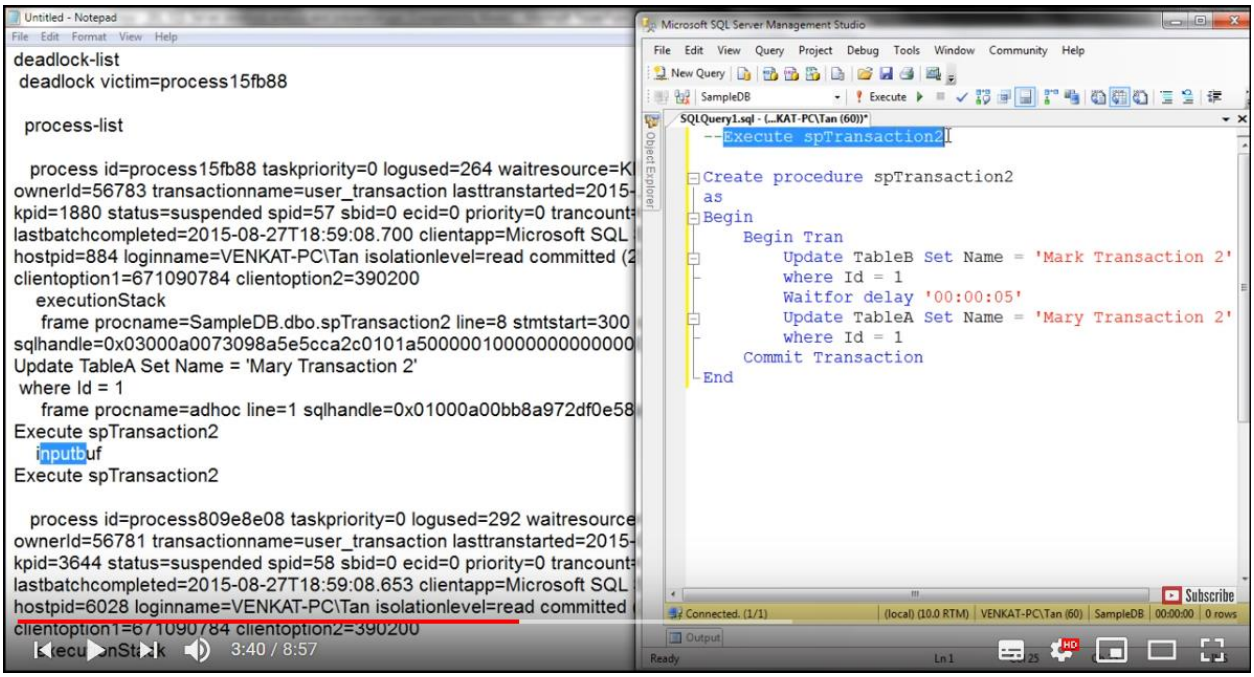
When we compare the Process ID of the victim with the process List then we find that it matches the first process id. This means that this is the process that is made the deadlock victim. So whatever transaction this process was executing is rolled back. Accordingly, spTransaction2 is made the deadlock victim.

We can also see the loginname and isolation level etc.

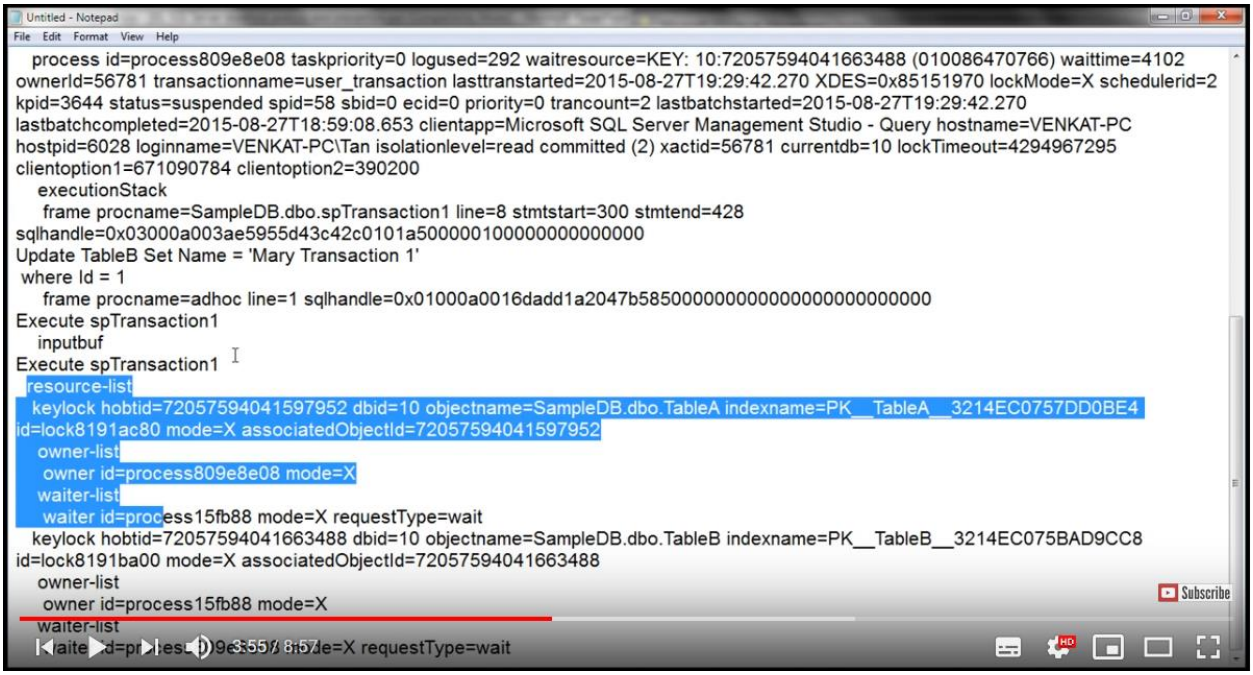
We can also see the fully qualified procedure name. In the notepad when we look at the procname that caused the deadlock, we can see the fully qualified name of the stored procedure.



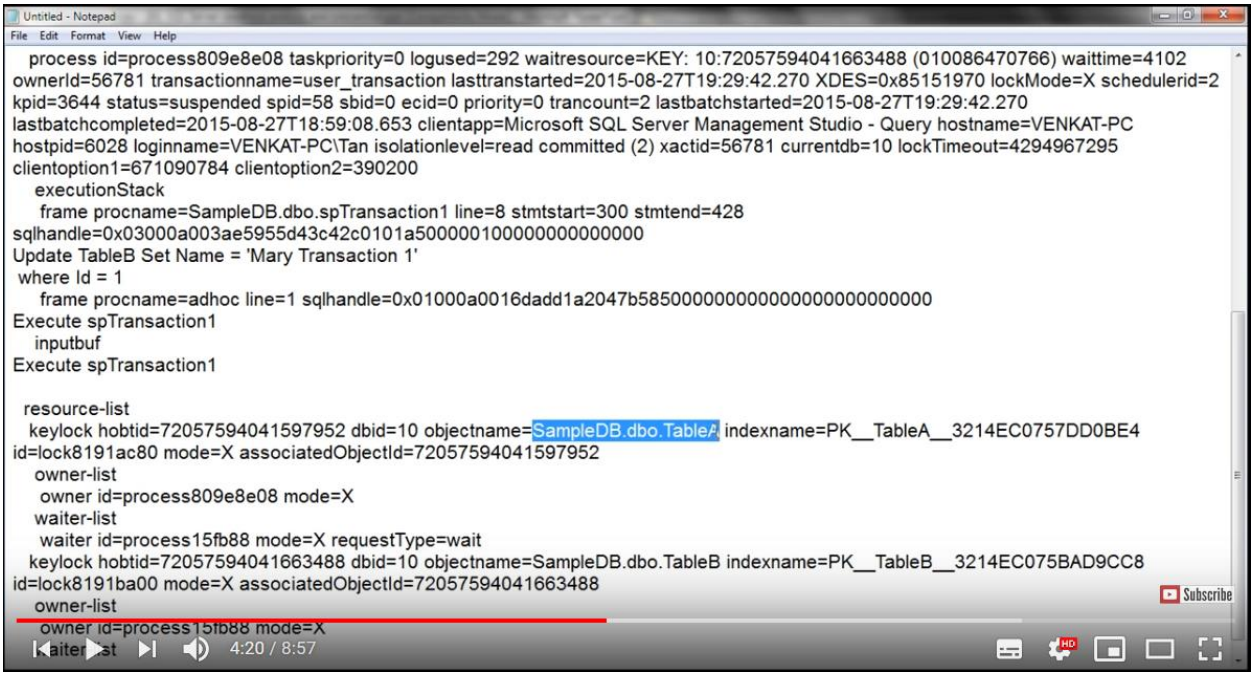
In the process list section we see that Inputbuf contains the code the process is executing:



The next very important section is the resource list section. This section contains list of resources i.e. objects owned by the processes involved in the deadlock.



The important thing that we note here is the object name, and that resource has an owner process and a waiting process.



If we analyse the log of owner process and waiting process then we come to conclusion that no one can move forward. So, how to solve this problem?

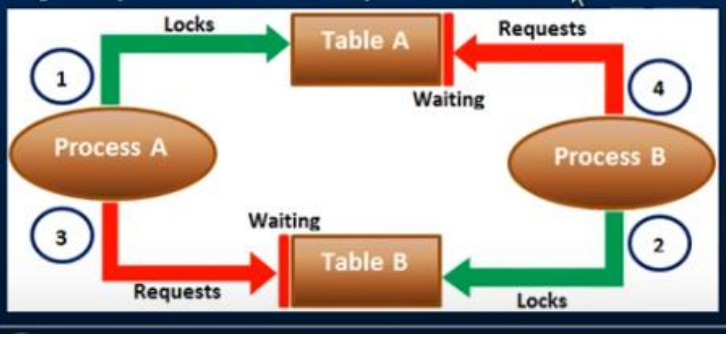
Straightforward, we only need to ensure that the database objects (TableA and TableB) are always accessed in the same order.

Deadlock analysis and prevention

Resource List Section

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



So, let us change the order from:

```
--Execute spTransaction1

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
```

```
--Execute spTransaction2

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
```

to

```
--Execute spTransaction1

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
```

```
--Execute spTransaction2

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

Now, in both cases, tables are being accessed in same order.

IF WE EXECUTE THE PROCEDURE SO THAT TABLES ARE UPDATED IN SAME ORDER, THAT IS BOTH TABLE A AND TABLE B GETS UPDATED IN SAME ORDER IN BOTH TRANSACTIONS: Both Transactions Succeed:

```
Execute spTransaction1

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
```

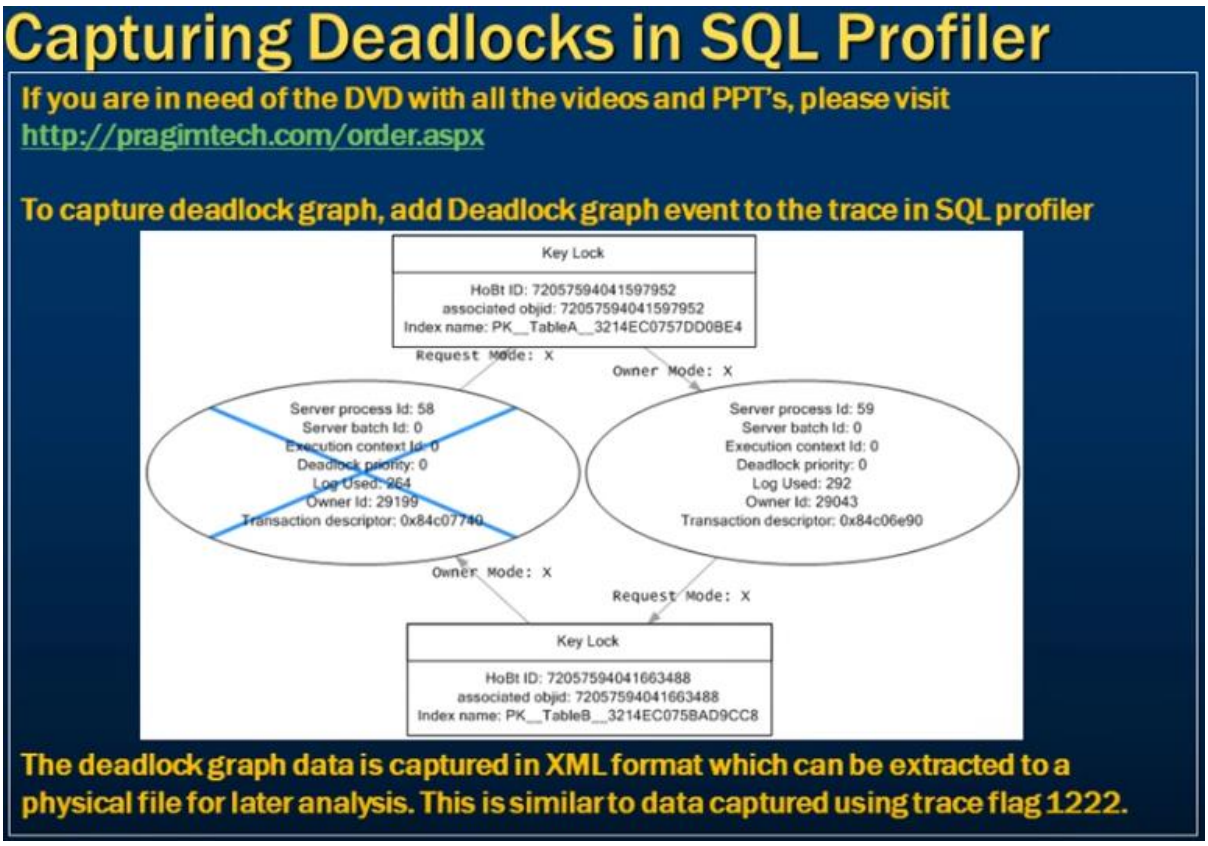
```
(1 row(s) affected)

(1 row(s) affected)
```

```
Execute spTransaction2

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

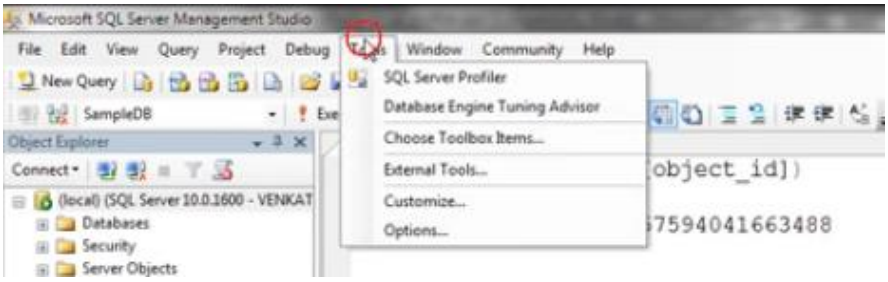
```
(1 row(s) affected)

(1 row(s) affected)
```

Capturing deadlocks

- How to capture deadlock graph using SQL profiler

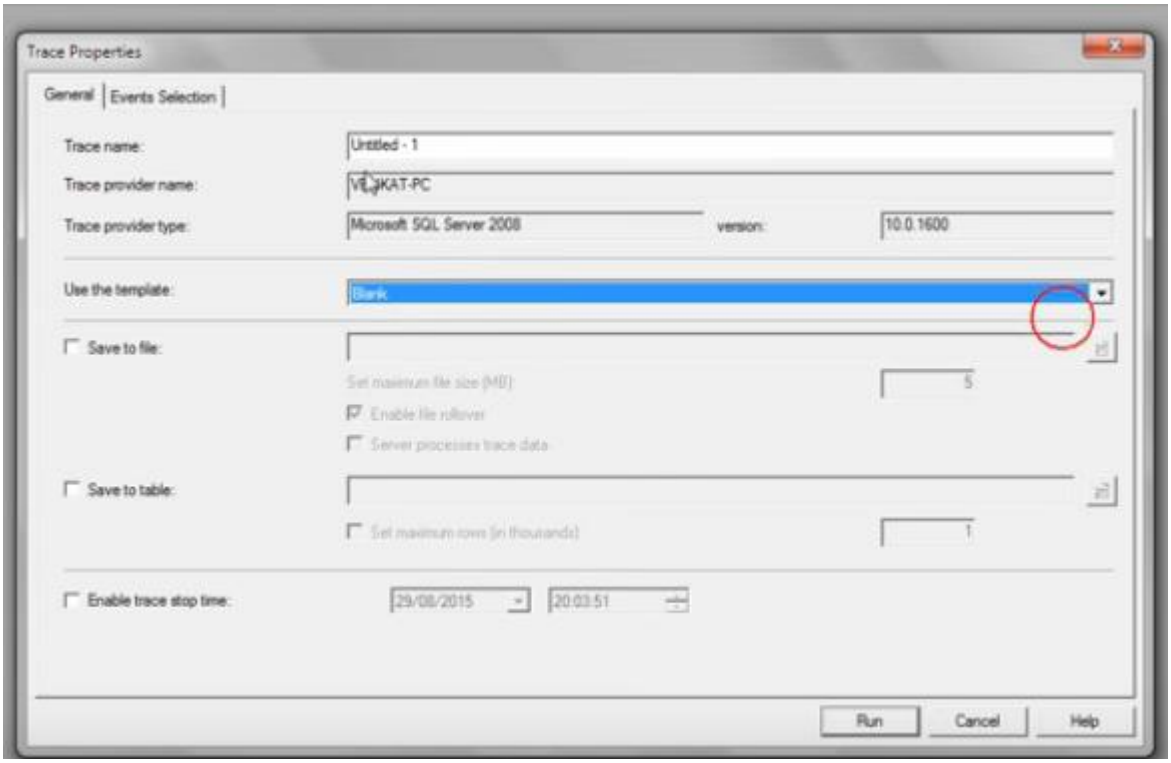


GO TO TOOLS -> SQL SERVER PROFILER

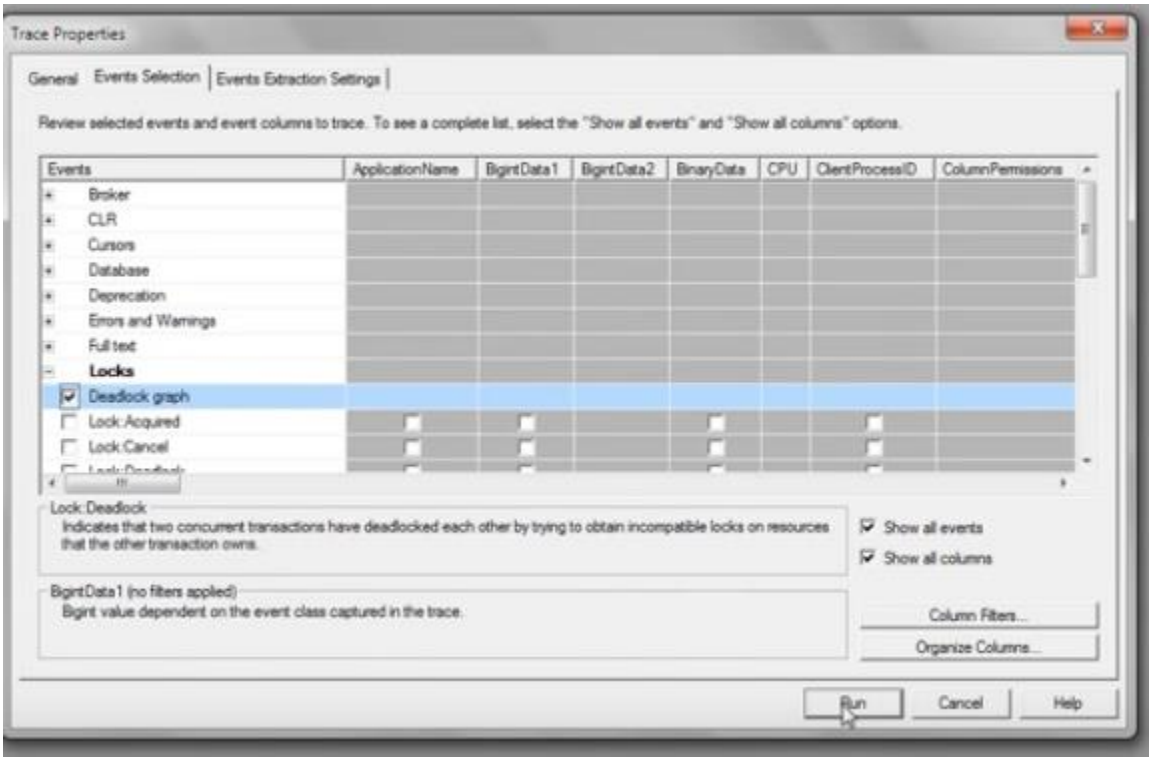


We will have to provide name of SQL Server.

ON TRACE PROPERTIES WINDOW, SELECT THE BLANK TEMPLATE:

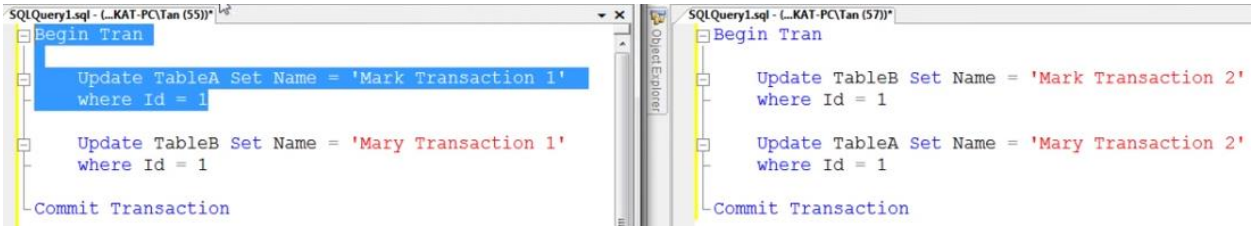


IN LOCKS SECTION, SELECT DEADLOCK GRAPH:



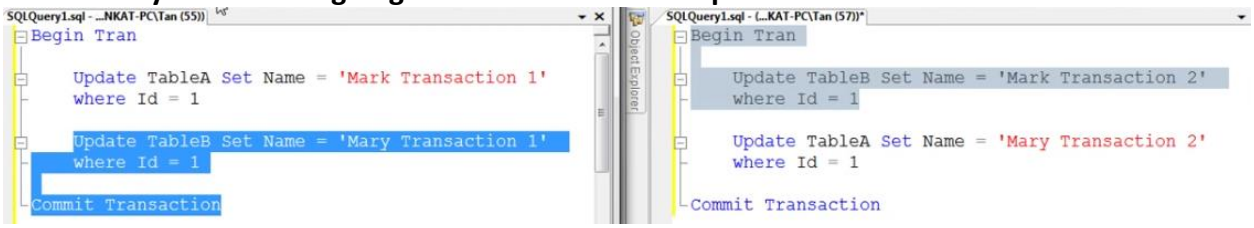
Click Run button to start a new trace. If now a deadlock occurs it will be captured.

We have two instances of SSMS running and two transactions. We execute part of our first transaction.

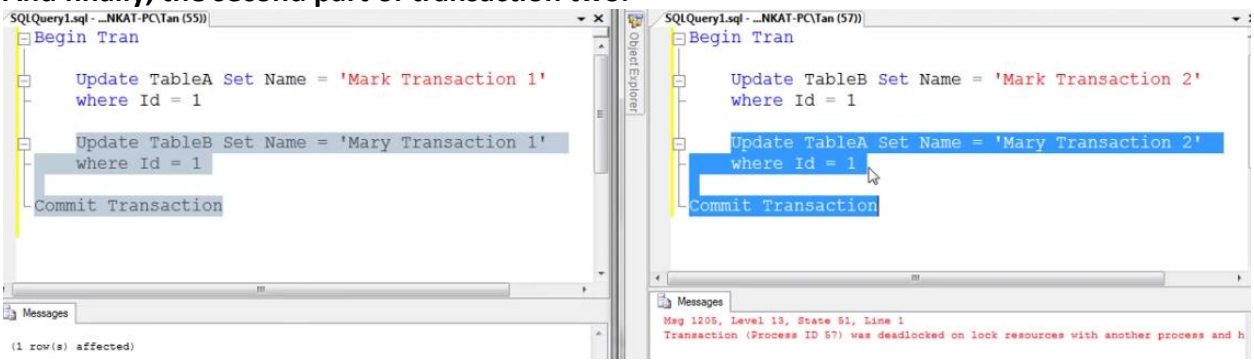


And then part of our second transaction.

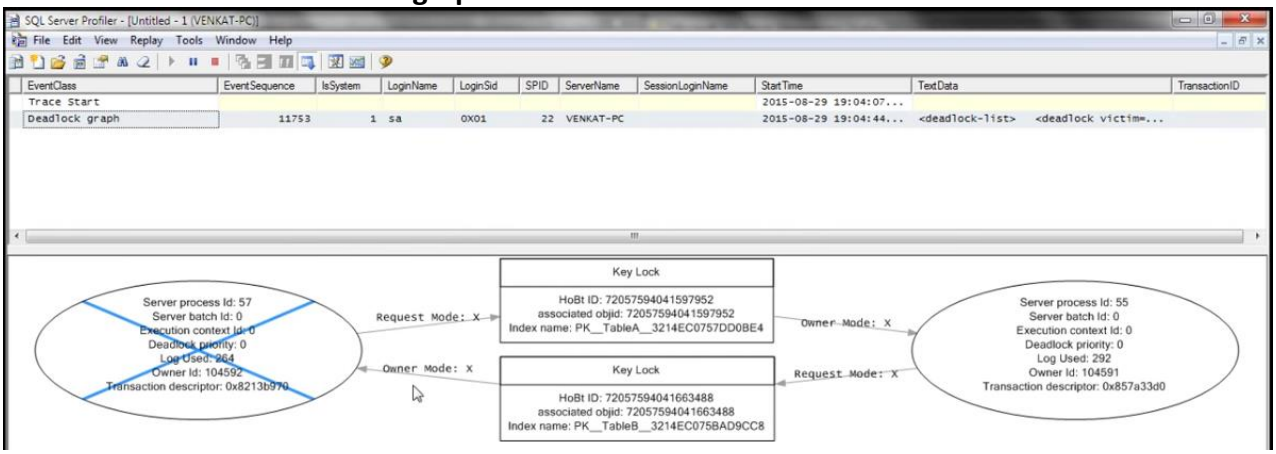
Followed by that we are going to execute the second part of first transaction.



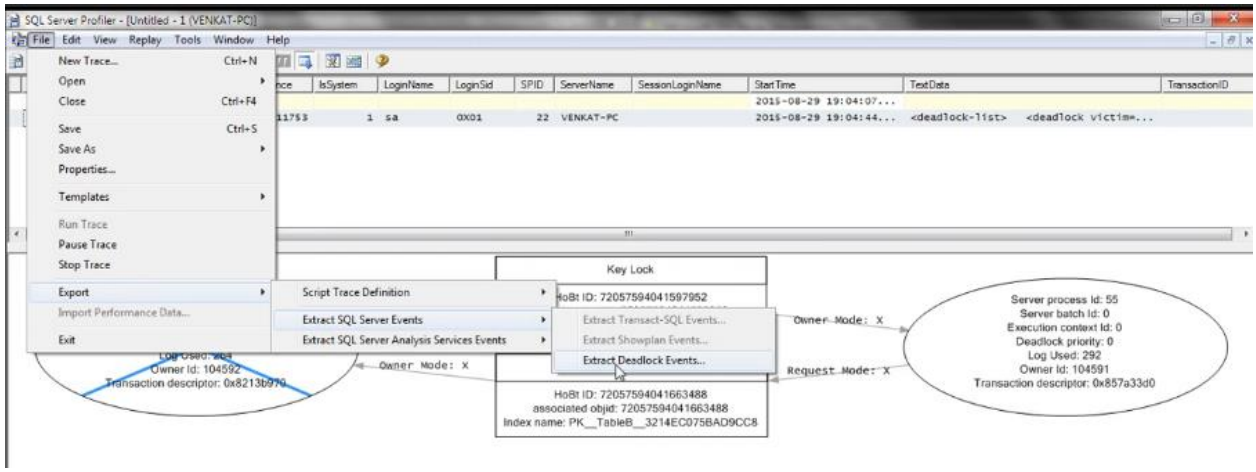
And finally, the second part of transaction two.



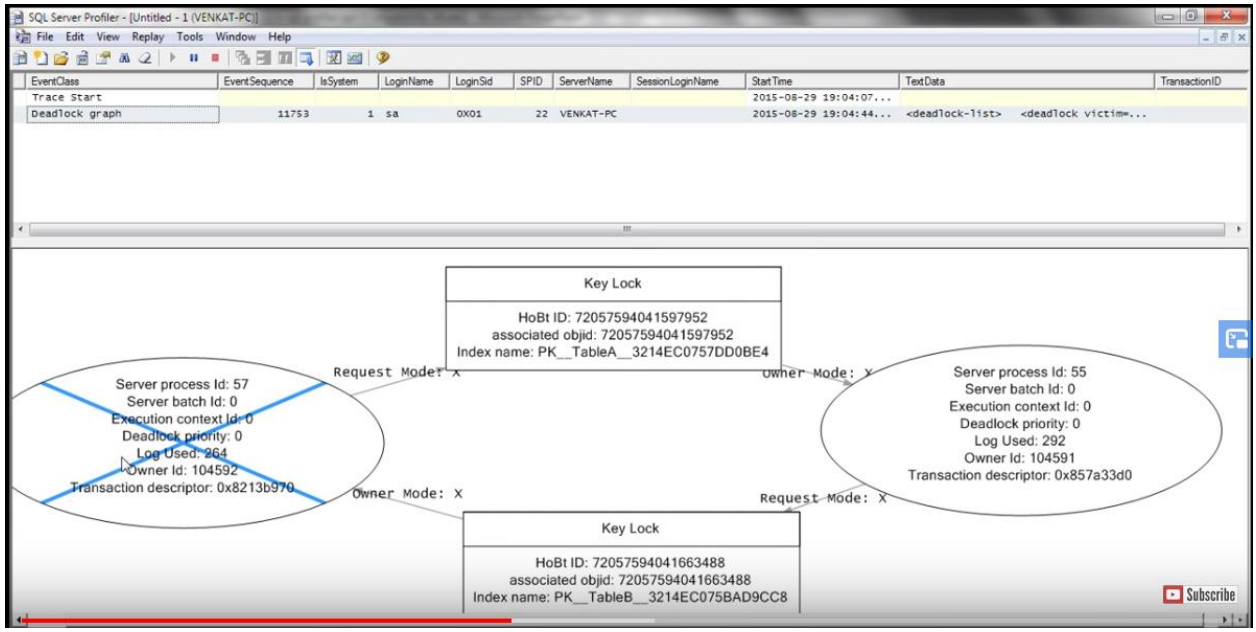
We land up in a deadlock. The information about this deadlock is captured by SQL Profiler and we have deadlock graph now.



THE UNDERLYING DATA ABOUT DEADLOCK IS IN XML FORMAT. TO EXTRACT THE XML DATA INTO A PHYSICAL FILE, WE NEED TO GO TO FILE-> EXPORT -> EXTRACT SQL SERVER EVENTS -> EXTRACT DEADLOCK EVENTS



We have two ovals and two rectangles. The oval represent the processes. THE OVAL WITH BLUE CROSS IS THE DEADLOCK VICTIM. The other oval is the transaction that completed successfully.



THE ARROW WITH OWNER MODE SHOWS WHICH OBJECT THE PARTICLUAR PROCESS HAS PUT AN EXCLUSIVE LOCK ON. WE CAN SEE THE DEADLOCK VICTIM PROCESS WITH BLUE CROSS HAS LOCKED THE TABLE B AND IS WAITING FOR TABLE A IN REQUEST MODE.

Analyzing 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 without blue cross represents the transaction that completed successfully.

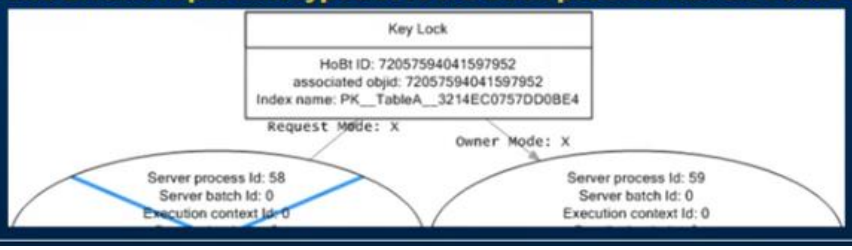
3. When you move the mouse pointer over the oval, you can see the SQL code that was running that caused the deadlock.

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.
   a) 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.



THIS IS HOW WE CHECK THE OBJECT INVOLVED IN DEADLOCK USING HOBT_ID

Deadlock error handling

Let's say we have two stored procedures and both try to update two tables. If we execute these two together we land up in deadlock.

```
Execute spTransaction1

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
```

```
Execute spTransaction2

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
```

Just like other errors, deadlock errors can be handled using Try/Catch block.

```
Execute spTransaction1

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
        Commit Transaction
        Select 'Transaction Successful!'
    End Try
    Begin Catch
        if(ERROR_NUMBER() = 1205)
        Begin
            Select 'Deadlock. Transaction Failed. Ple
        End
        Rollback
    End Catch
End
```

```
Execute spTransaction2

Alter procedure spTransaction2
as
Begin
    Begin Tran
    Begin Try
        Update TableB Set Name = 'Mary Transaction 2'
        Waitfor delay '00:00:05'
        Update TableA Set Name = 'Mark Transaction 2'
        Commit Transaction
        Select 'Transaction Successful'
    End Try
    Begin Catch
        If(ERROR_NUMBER() = 1205)
        Begin
            Select 'Deadlock. Transaction failed. Ple
        End
        Rollback
    End Catch
End
```

While the first transaction will succeed

Results | Messages

	(No column name)
1	Transaction Successful!

When we run the second transaction it will cause deadlock and we will get message:

Results | Messages

	(No column name)
1	Deadlock. Transaction failed. Please retry

How to find blocking queries

How to find blocking queries in sql server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Blocking occurs if there are open transactions

DBCC OpenTran will display only the oldest active transaction. It is not going to show you all the open transactions

SQL Server process can be killed using

1. SQL Server Activity Monitor
2. 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

Let's say we have two instances of SSMS and a transaction in first instance is blocking the transactions in the second instance because we have no executed commit, then how to find the blocking transaction.

```
SQLQuery4.sql - (...KAT-PC\Tan (54))*
Begin Tran
Update TableA set Name='Mark Transaction 1'
where Id = 1

Commit
```

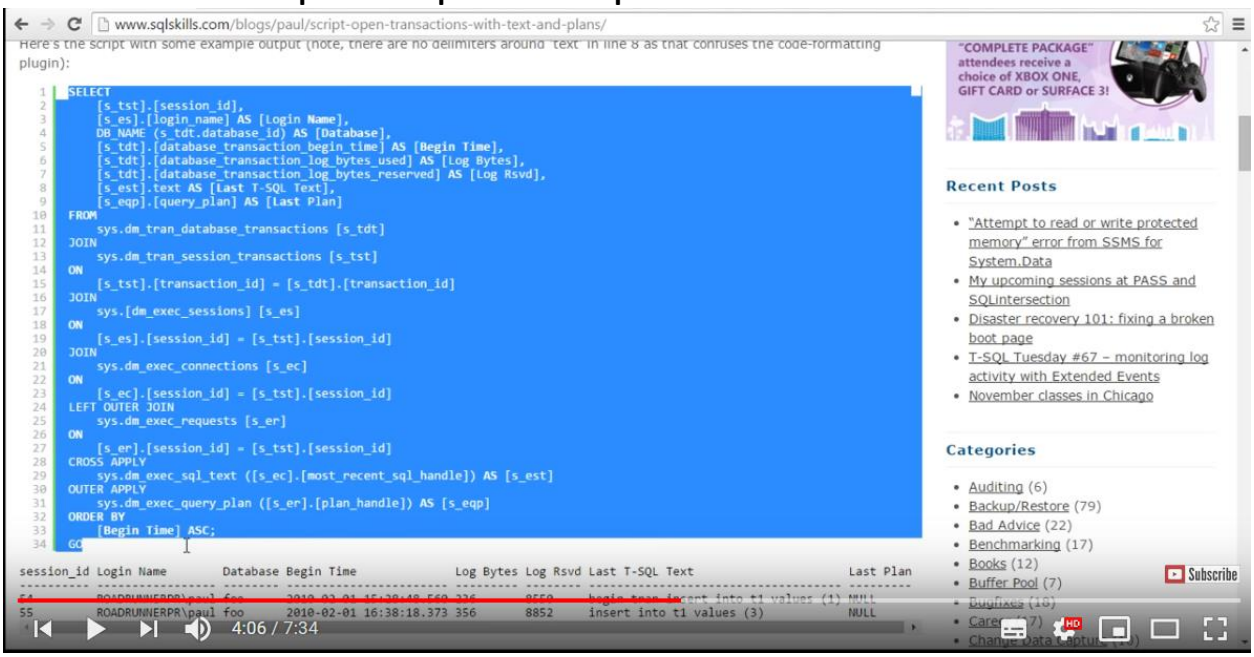
```
SQLQuery3.sql - (...KAT-PC\Tan (58))*
Select Count(*) from TableA

Delete from TableA where Id = 1

Truncate table TableA

Drop table TableA
```

Here, we are aware that our transaction in the first instance of SSMS is the one causing the blocking but in real world scenario with multiple users we will not know who or which transaction is causing the blocking. But most likely it will be due to an open transaction. One way to find this is by using DBCCOpenTran command.

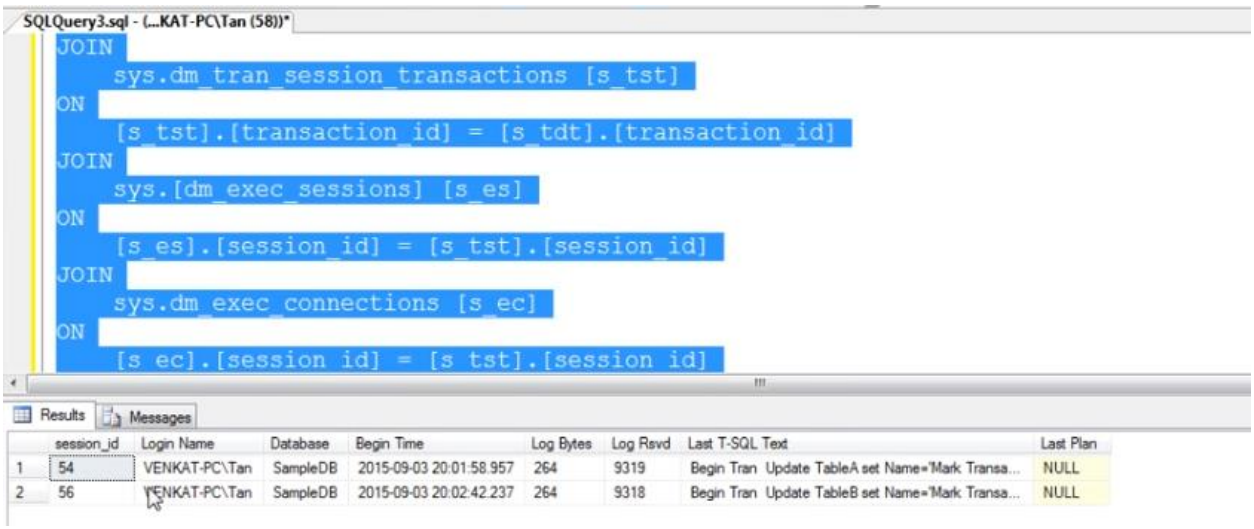
```
DBCC OpenTran
```

```
Messages
Transaction information for database 'SampleDB'.

Oldest active transaction:
    SPID (server process ID): 54
    UID (user ID) : -1
    Name          : user_transaction
    LSN           : (762:115:1)
    Start time    : Sep  3 2015  8:01:58:957PM
    SID           : 0x0105000000000000051500000061d6d7b5cafb90bee99abd85e8030000
DBCC execution completed. If DBCC printed error messages, contact your system adminis
```

Is it possible to list all the open active transactions? Yes, on the web on sqlskills.com there is a valuable script that helps us list all open active transactions.

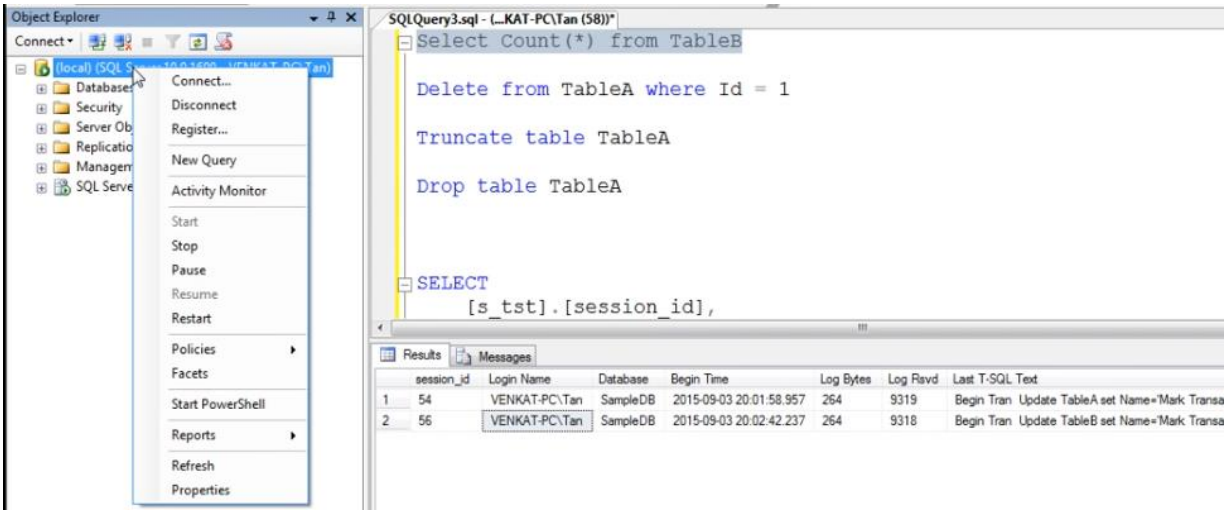


Let's copy the script, paste in query window, and execute the query. It gives lots more information about all open transactions.
It give us the session_id, Login name, Database name, and lots more information.
The beauty is that it also has the SQL script text.

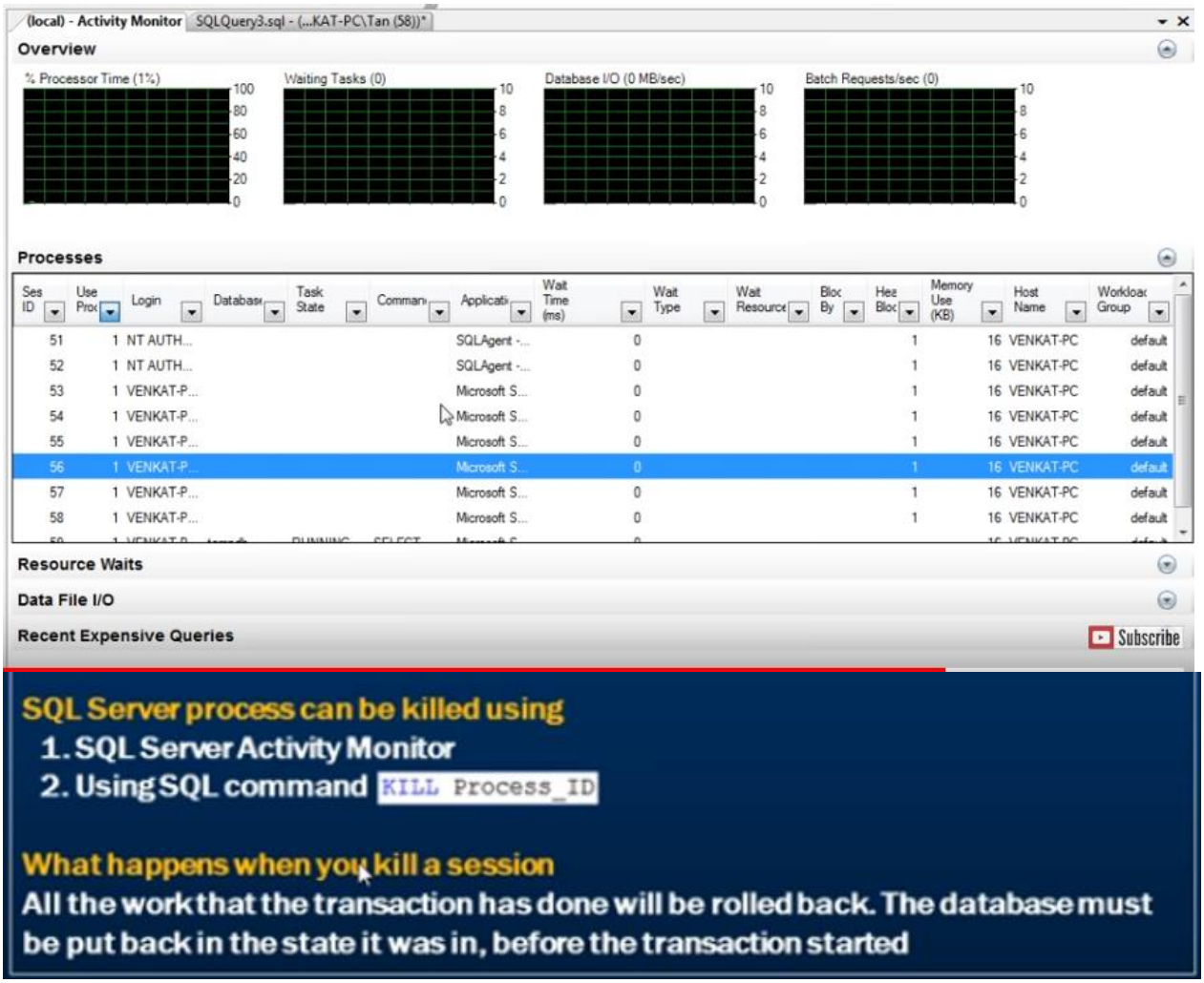


Once we know about a process causing problem, we can kill it. There are two ways to do that.

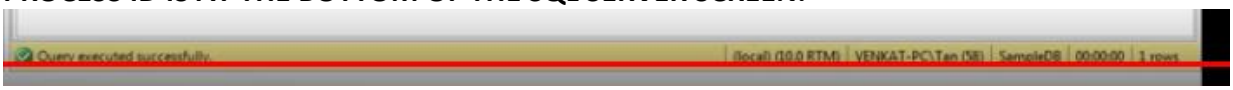
TO KILL A PROCESS RIGHT CLICK ON LOCAL-> CLICK ACTIVITY MONITOR



GO TO PROCESSES AND RIGHT CLICK THE PROCESS NUMBER TO KILL PROCESS.

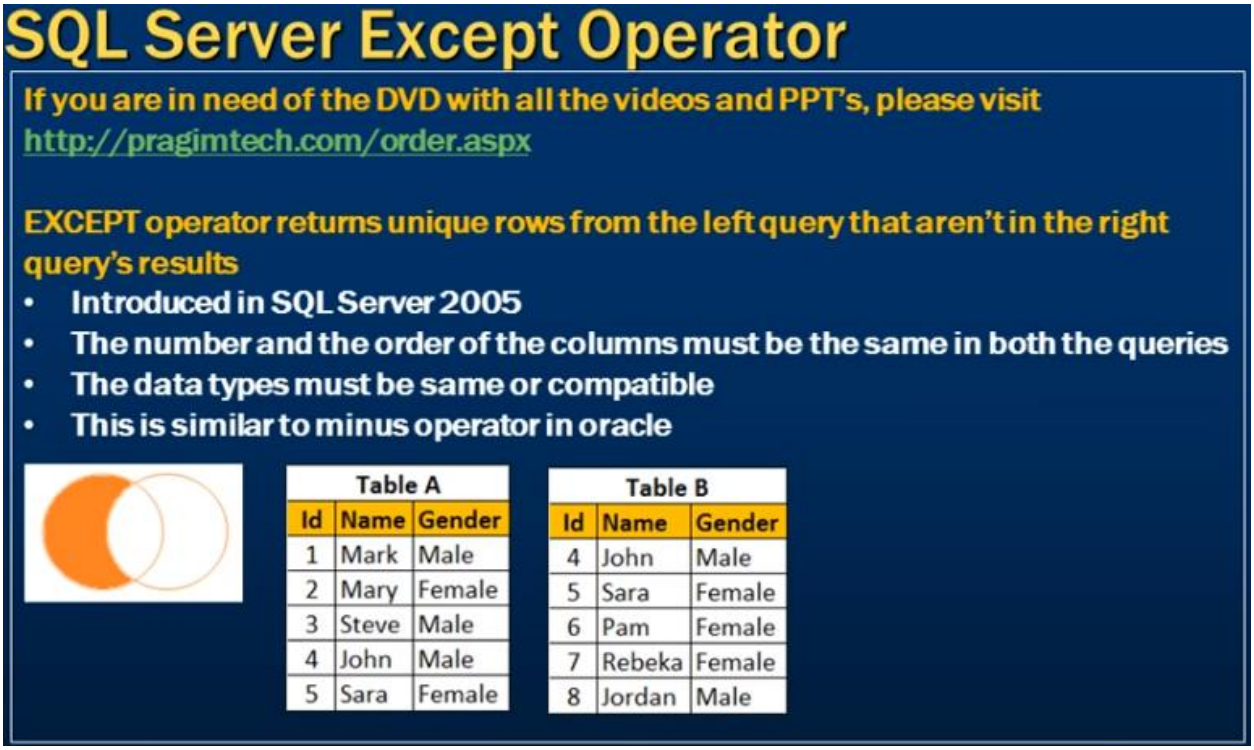


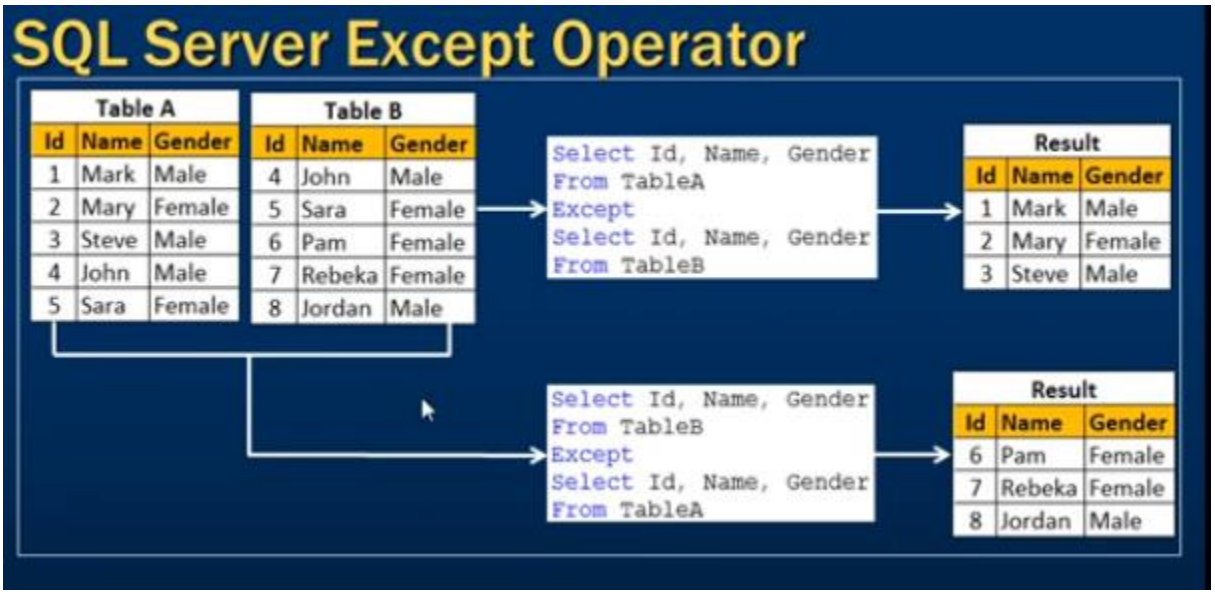
PROCESS ID IS AT THE BOTTOM OF THE SQL SERVER SCREEN:



IN ABOVE PC ID IS 58

Except operator





EXCEPT OPERATOR CAN ALSO BE USED ON SINGLE TABLE.

THE BELOW STATEMENTS SELECTS ALL EMPLOYEES WHOSE SALARY IS BETWEEN 50000 AND 60000.

SQL Server Except Operator

You can also use Except operator on a single table

tblEmployees

Id	Name	Gender	Salary
1	Mark	Male	52000
2	Mary	Female	55000
3	Steve	Male	45000
4	John	Male	40000
5	Sara	Female	48000
6	Pam	Female	60000
7	Tom	Male	58000
8	George	Male	65000
9	Tina	Female	67000
10	Ben	Male	80000

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

Result

Id	Name	Gender	Salary
1	Mark	Male	52000
2	Mary	Female	55000
7	Tom	Male	58000

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 Id Desc
```

Result

Id	Name	Gender	Salary
7	Tom	Male	58000
2	Mary	Female	55000
1	Mark	Male	52000

SQLQuery3.sql - (...KAT-PC\Tan (56))* | SQLQuery2.sql - (...KAT-PC\Tan (55))* | SQLQuery1.sql - (...KAT-PC\Tan (54))*

```
Select Id, Name, Gender, Salary from tblEmployees where Salary >= 50000
Except
Select Id, Name, Gender, Salary from tblEmployees where Salary >= 60000
```

Results | Messages

	Id	Name	Gender	Salary
1	1	Mark	Male	52000
2	2	Mary	Female	55000
3	7	Tom	Male	58000

CAN ALSO USE ORDER BY CLAUSE BUT IT MUST BE USED AFTER THE LAST SELECT QUERY.

Difference between except and not in

Difference between EXCEPT and NOT IN

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

EXCEPT operator returns all the rows from the left query that aren't in the right query's results. NOT IN operator also does the same.

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male

Table B

Id	Name	Gender
2	Mary	Female
3	Steve	Male

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

```
Select Id, Name, Gender From TableA
Where Id NOT IN
(Select Id from TableB)
```

Result

Id	Name	Gender
1	Mark	Male

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

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 sub-query

IF WE INSERT AN EXTRA RECORD FOR MARK IN TABLE A, THEN WE GET DUPLICATES IN THE CASE OF ''NOT IN'' OPERATOR BUT NOT IN THE CASE OF EXCEPT OPERATOR.

```
--Insert into TableA values (1, 'Mark', 'Male')

Select Id, Name, Gender from TableA
Except
Select Id, Name, Gender from TableB

Select Id, Name, Gender from TableA
where Id NOT IN (Select Id from TableB)
```

Results | Messages

Id	Name	Gender
1	Mark	Male
1	Mark	Male

Intersect operator

- INTERSECT operator in SQL Server
- Difference between INTERSECT & INNER JOIN

INTERSECT Operator in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

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

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male

Table B

Id	Name	Gender
2	Mary	Female
3	Steve	Male

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

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

Result

Id	Name	Gender
2	Mary	Female
3	Steve	Male

INTERSECT v/s INNER JOIN

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 make INNER JOIN behave like INTERSECT operator use the DISTINCT operator

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

```
Select Id, Name, Gender from TableA

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

Select DISTINCT TableA.Id, TableA.Name, TableA.Gender from TableA
INNER JOIN TableB
ON TableA.Id = TableB.Id
```

	Id	Name	Gender
1	2	Mary	Female
2	3	Steve	Male

	Id	Name	Gender
1	2	Mary	Female
2	3	Steve	Male

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

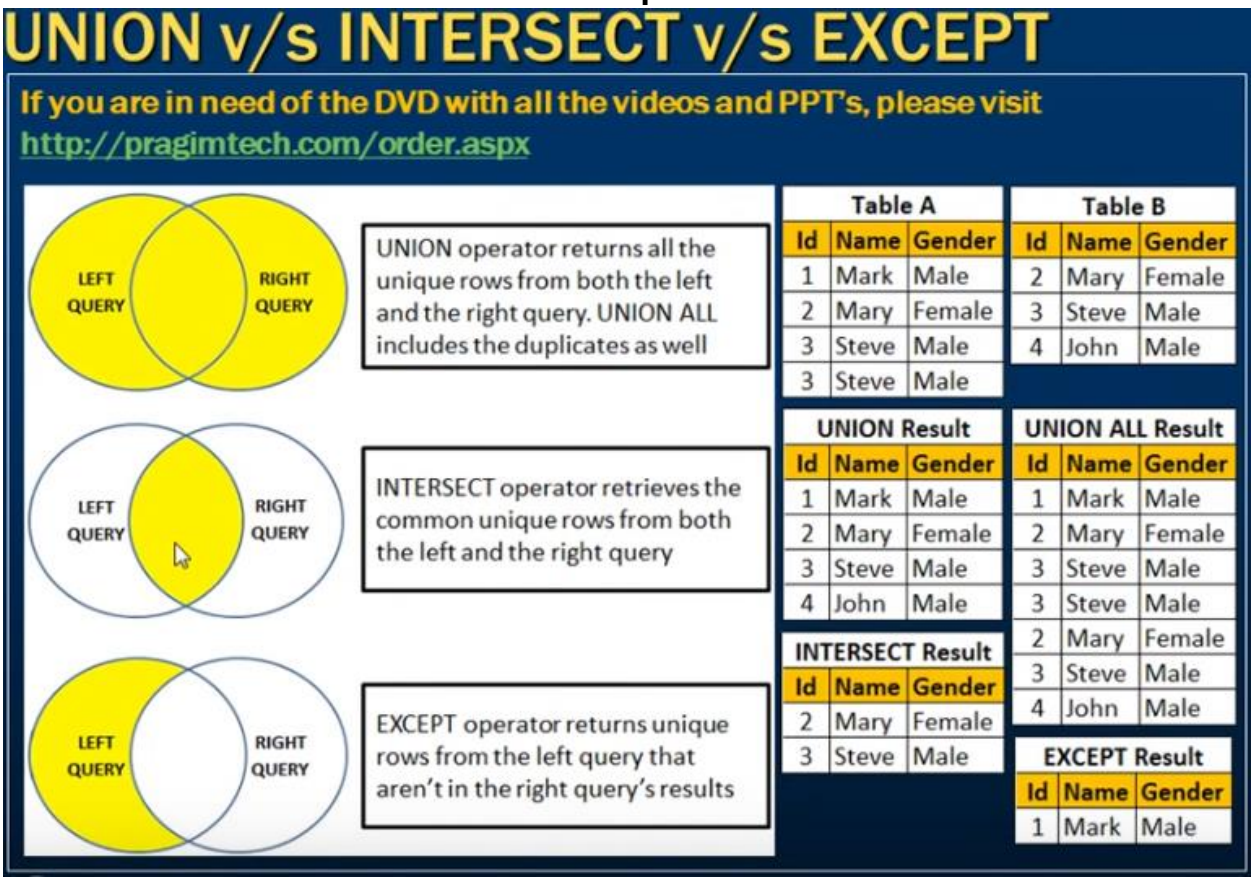
```
Select Id, Name, Gender from TableA
Select Id, Name, Gender from TableB

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

Select DISTINCT TableA.Id, TableA.Name, TableA.Gender from TableA
INNER JOIN TableB
ON TableA.Id = TableB.Id
```

	Id	Name	Gender
1	NULL	Pam	Female
2	2	Mary	Female
3	3	Steve	Male

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

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

Select DISTINCT TableA.Id, TableA.Name, TableA.Gender from TableA
INNER JOIN TableB
ON TableA.Id = TableB.Id
```

	Id	Name	Gender
1	2	Mary	Female
2	3	Steve	Male

Difference between union intersect and except




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

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

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

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

Cross apply and outer apply

Cross apply & Outer apply in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Department Table

Id	DepartmentName
1	IT
2	HR
3	Payroll
4	Administration
5	Sales

Employee Table

Id	Name	Gender	Salary	DepartmentId
1	Mark	Male	50000	1
2	Mary	Female	60000	3
3	Steve	Male	45000	2
4	John	Male	56000	1
5	Sara	Female	39000	2

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
Payroll	Mary	Female	60000
HR	Steve	Male	45000
IT	John	Male	56000
HR	Sara	Female	39000

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
IT	John	Male	56000
HR	Steve	Male	45000
HR	Sara	Female	39000
Payroll	Mary	Female	60000
Administration	NULL	NULL	NULL
Sales	NULL	NULL	NULL

- 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

SQLQuery1.sql - (...KAT-PC\Tan (54))* SQLQuery2.sql - (...KAT-PC\Tan (55))*

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

Results Messages

	DepartmentName	Name	Gender	Salary
1	IT	Mark	Male	50000
2	Payroll	Mary	Female	60000
3	HR	Steve	Male	45000
4	IT	John	Male	56000
5	HR	Sara	Female	39000

- 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)

```
Create function fn_GetEmployeesByDepartmentId(@DepartmentId int)
returns table
as
Return
(
    Select * from Employee
    where DepartmentId = @DepartmentId
)

Select * from fn_GetEmployeesByDepartmentId(1)
```

SQLQuery1.sql - (...KAT-PC\Tan (54))* | SQLQuery2.sql - (...KAT-PC\Tan (55))*

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

Results | Messages

	DepartmentName	Name	Gender	Salary
1	IT	Mark	Male	50000
2	Payroll	Mary	Female	60000
3	HR	Steve	Male	45000
4	IT	John	Male	56000
5	HR	Sara	Female	39000

- 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

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

Results | Messages

	DepartmentName	Name	Gender	Salary
1	IT	Mark	Male	50000
2	IT	John	Male	56000
3	HR	Steve	Male	45000
4	HR	Sara	Female	39000
5	Payroll	Mary	Female	60000
6	Administration	NULL	NULL	NULL
7	Sales	NULL	NULL	NULL

DDL Triggers

DDL Triggers in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

In SQL Server there are 4 types of triggers

1. DML Triggers - Data Manipulation Language. Parts 43 to 47 (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

DDL Triggers in SQL Server

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

DDL trigger Syntax

```
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

DDL Triggers Example

This trigger fires whenever a table is created

```
-- Trigger that fires in response to a single DDL event
CREATE TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE
AS
BEGIN
   Print 'New table created'
END
```

This trigger fires whenever a table is created, altered or dropped

```
-- Trigger that fires in response to a multiple DDL
-- events
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   Print 'You just created, modified or deleted a table'
END
```

SQLQuery1.sql - (...KAT-PC\Tan (54))*

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

Create Table Test (Id int)
```

Messages

Command(s) completed successfully.

The above examples are executing some code in response to some DDL events. Another use is of triggers is to prevent changes to Schema.

Whenever someone is trying to create, alter, or drop a table, we can do so using Rollback statement whenever

DDL Triggers in SQL Server

```
-- DDL trigger to prevent users from creating,
-- altering or dropping tables
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
```

Now when Create table is executed, we get cannot create message.

SQLQuery1.sql - (...KAT-PC\Tan (54))*

```
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


Create Table Test (Id int)
Drop Table Test
```

Messages

```
You cannot create, alter or drop a table
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

We can disable a trigger as follows:

```
Disable Trigger trMyFirstTrigger ON Database
```

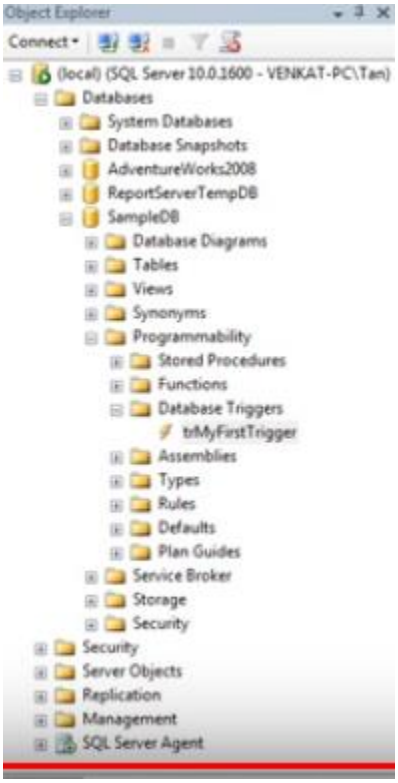
Messages

```
Command(s) completed successfully.
```

We can also drop a trigger:

```
DROP Trigger trMyFirstTrigger ON Database
```

CAN CHECK LIST OF DATABASE TRIGGERS IN OBJECT EXPLORER UNDER THE CURRENT DATABASE:



THE FOLLOWING TRIGGER WILL NOT ALLOW TO CREATE ANY NEW TABLE DUE TO ROLLBACK IN THE DDL TRIGGER

```
SQLQuery1.sql - (...KAT-PC\Tan (54))*
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


Create Table Test (Id int)
Drop Table Test
```

TO DISABLE A TRIGGER, WE NEED TO USE:

```
Disable Trigger trMyFirstTrigger ON Database
```

TO ENABLE OR DROP TRIGGER USE: 'ENABLE TRIGGER' OR 'DROP TRIGGER'

DDL Triggers in SQL Server

Certain system stored procedures that perform DDL-like operations can also fire DDL triggers

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

The trigger will be fired when ever you rename a database object using sp_rename system stored procedure

```
-- Renaming table - Fires the trigger
sp_rename 'TestTable', 'NewTestTable'

-- Renaming table column - Fires the trigger
sp_rename 'NewTestTable.Id' , 'NewId', 'column'
```

Server scoped DDL triggers

Server-Scoped DDL Triggers

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

```
-- Database scoped trigger
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
```

```
-- Server scoped trigger
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
```

Database scoped trigger:

```
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'
END
```

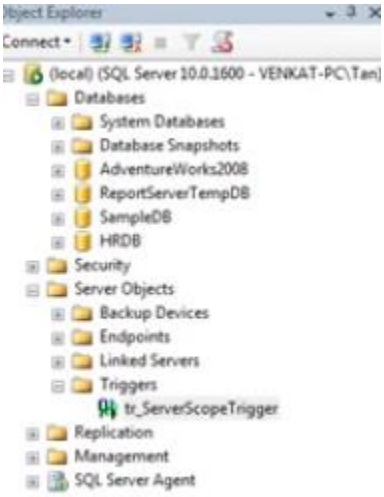
CREATING A TRIGGER FOR ENTIRE SERVER FOR ALL THE DATABASES ON THE SERVER:

```
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'
END

-- Create Table
Create Table Test(Id int)
```

Messages

Command(s) completed successfully.

CAN LOCATE SERVER LEVEL TRIGGERS UNDER SERVER OBJECTS:



TO DISABLE A SERVER LEVEL TRIGGER:

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

Server scoped triggers will always fire before any of the database scoped triggers

WHEN WE HAVE MULTIPLE DATABASE TRIGGERS, THEY FIRE IN THE SAME ORDER AS THEY WERE CREATED:

```
CREATE TRIGGER tr_DatabaseScopeTrigger3
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    Print 'Database Scope Trigger - 3'
END
GO

CREATE TRIGGER tr_DatabaseScopeTrigger2
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    Print 'Database Scope Trigger - 2'
END
GO

CREATE TRIGGER tr_DatabaseScopeTrigger1
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    Print 'Database Scope Trigger - 1'
```

Trigger execution order

SQL Server Trigger Execution Order

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Server scoped triggers will always fire before any of the database scoped triggers

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

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
```

CAN CHANGE THE DEFAULT ORDER OF EXECUTION OF TRIGGER USING SP_TRIGGERORDER

```
EXEC sp_settriggerorder
@triggername = 'tr_DatabaseScopeTrigger1',
@order = 'first',
@stmttype = 'CREATE_TABLE',
@namespace = 'DATABASE'
GO

EXEC sp_settriggerorder
@triggername = 'tr_DatabaseScopeTrigger3',
@order = 'last',
@stmttype = 'CREATE_TABLE',
@namespace = 'DATABASE'
GO
```

SQL Server Trigger Execution Order

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

Audit table changes

- How to audit table changes in SQL Server using a DDL trigger

We want to keep track of all table changes.

Audit Table Changes in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

DatabaseName	TableName	EventType	LoginName	SQLCommand	AuditDateTime
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:02:24.240
SampleDB	MyTable	DROP_TABLE	VENKAT-PC\Tan	Drop Table MyTabl	2015-09-11 16:02:57.430
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:06:07.037
SampleDB	MyTable	ALTER_TABLE	VENKAT-PC\Tan	Alter Table MyTabl	2015-09-11 16:06:11.610

EventData() function returns event data in XML format

```
<EVENT_INSTANCE>
  <EventType>CREATE_TABLE</EventType>
  <PostTime>2015-09-11T16:12:49.417</PostTime>
  <ServerName>VENKAT-PC</ServerName>
  <LoginName>VENKAT-PC\Tan</LoginName>
  <DatabaseName>SampleDB</DatabaseName>
  <ObjectName>MyTable</ObjectName>
  <TSQLCommand>
    <CommandText>
      Create Table MyTable(Id int,Name nvarchar(50),Gender nvarchar(50))
    </CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

```
CREATE TRIGGER tr_AuditTableChanges
ON ALL SERVER
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    Select EventData()
END
```

ENTERING INFORMATION IN TO AUDIT TABLE FROM THE XML DATA OF EVENTDATA() FUNCTION:

```
<EVENT_INSTANCE>
  <EventType>CREATE_TABLE</EventType>
  <PostTime>2015-09-11T19:28:32.393</PostTime>
  <SPID>54</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"
    <CommandText>Create Table MyTable (Id int, Name nvarchar(50))</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

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

Create Table MyTable (Id int, Name nvarchar(50))

Alter Table MyTable
Alter Column Name nvarchar(100)

Drop Table MyTable
```

```sql
ALTER 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
```

RESULT:

Results | Messages

	DatabaseName	TableName	EventType	LoginName	SQLCommand	AuditDateTime
1	SampleDB	MyTable	ALTER_TABLE	VENKAT-PC\Tan	Alter Table MyTable Alter Column Name nvarchar(...	2015-09-11 19:32:04.730
2	SampleDB	MyTable	DROP_TABLE	VENKAT-PC\Tan	Drop Table MyTable	2015-09-11 19:32:10.000
3	HRDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTable (Id int, Name nvarchar(50))	2015-09-11 19:32:57.257

DatabaseName	TableName	EventType	LoginName	SQLCommand	AuditDateTime
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:02:24.240
SampleDB	MyTable	DROP_TABLE	VENKAT-PC\Tan	Drop Table MyTabl	2015-09-11 16:02:57.430
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:06:07.037
SampleDB	MyTable	ALTER_TABLE	VENKAT-PC\Tan	Alter Table MyTabl	2015-09-11 16:06:11.610

Logon Triggers

Logon Triggers in SQL Server

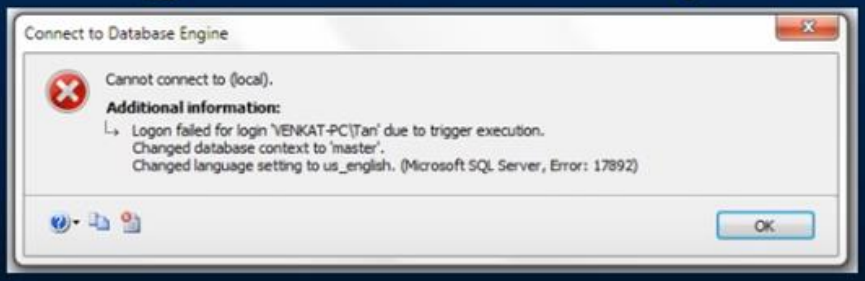
If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

As the name implies Logon triggers fire in response to a LOGON event. Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established

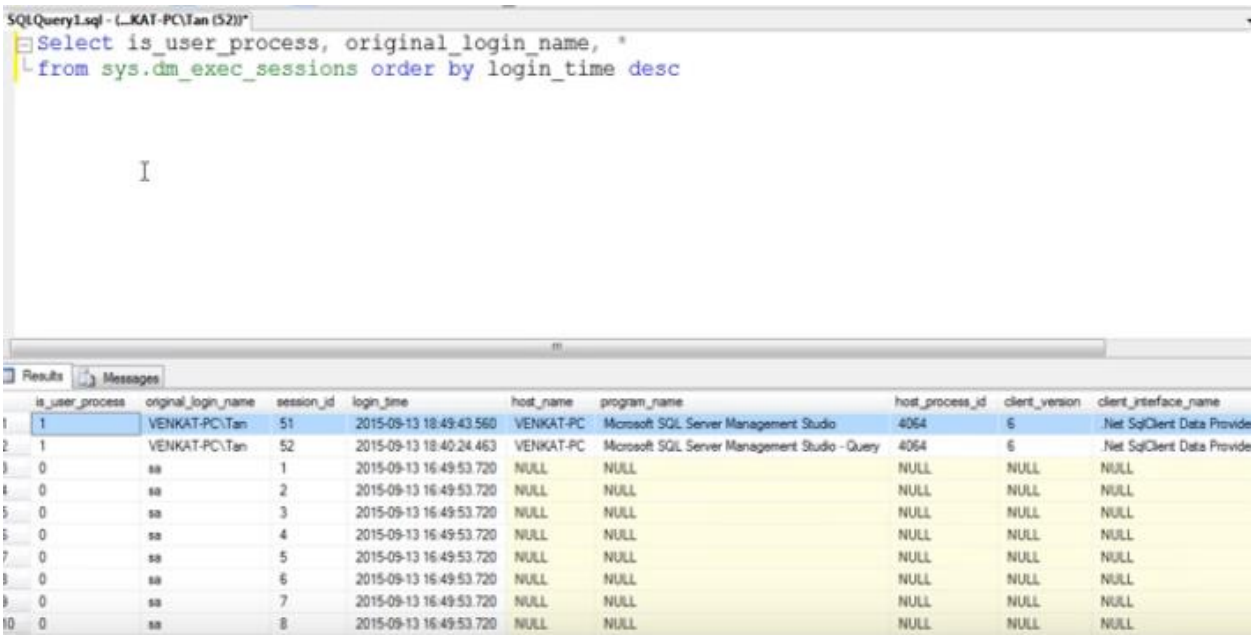
Logon triggers can be used for

1. Tracking login activity
2. Restricting logins to SQL Server
3. Limiting the number of sessions for a specific login

Write a trigger to limit the maximum number of open connections for a user to 3



Use the following query to check all the open connections:



	is_user_process	original_login_name	session_id	login_time	host_name	program_name	host_process_id	client_version	client_interface_name
1	1	VENKAT-PC\Tan	51	2015-09-13 18:49:43.560	VENKAT-PC	Microsoft SQL Server Management Studio	4064	6	.Net SqlClient Data Provide
2	1	VENKAT-PC\Tan	52	2015-09-13 18:40:24.463	VENKAT-PC	Microsoft SQL Server Management Studio - Query	4064	6	.Net SqlClient Data Provide
3	0	sa	1	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
4	0	sa	2	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
5	0	sa	3	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
6	0	sa	4	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
7	0	sa	5	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
8	0	sa	6	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
9	0	sa	7	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL
10	0	sa	8	2015-09-13 16:49:53.720	NULL	NULL	NULL	NULL	NULL

LIMITING THE NUMBER OF OPEN CONNECTIONS TO 3:

```
Create trigger tr_AuditLogin
ON ALL SERVER
FOR LOGON
AS
BEGIN
    Declare @LoginName nvarchar(100)
    Set @LoginName = Original_Login()

    IF(Select Count(*) from sys.dm_exec_sessions
    where is_user_process = 1 AND
    original_login_name = @LoginName) > 3
    BEGIN
        Print 'Fourth connection attempt by ' + @LoginName + ' blocked'
        ROLLBACK;
    END
END
```

Select into

SELECT INTO in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Departments Table

DepartmentId	DepartmentName
1	IT
2	HR
3	Payroll

Employees Table

Id	Name	Gender	Salary	DeptId
1	Mark	Male	50000	1
2	Sara	Female	65000	2
3	Mike	Male	48000	3
4	Pam	Female	70000	1
5	John	Male	55000	2

The SELECT INTO statement in SQL Server, selects data from one table and inserts it into a new table

SELECT INTO statement in SQL Server can do the following

1. Copy all rows and columns from an existing table into a new table. This is extremely useful when you want to make a backup copy of the existing table

```
SELECT * INTO EmployeesBackup FROM Employees
```

2. Copy all rows and columns from an existing table into a new table in an external database

```
SELECT * INTO HRDB.dbo.EmployeesBackup FROM Employees
```

3. Copy only selected columns into a new table

```
SELECT Id, Name, Gender INTO EmployeesBackup FROM Employees
```

4. Copy only selected rows into a new table

```
SELECT * INTO EmployeesBackup FROM Employees WHERE DeptId = 1
```

5. Copy columns from 2 or more table into a new table

```
SELECT * INTO EmployeesBackup FROM Employees
INNER JOIN Departments
ON Employees.DeptId = Departments.DepartmentId
```

6. Create a new table whose columns and data types match with an existing table

```
SELECT * INTO EmployeesBackup FROM Employees WHERE 1 <> 1
```

```
--Copy all rows and columns from an existing table into a new table
SELECT * INTO EmployeesBackup FROM Employees

SELECT * FROM EmployeesBackup
DROP TABLE EmployeesBackup

--Copy all rows and columns from an existing table into a new table in an external database
SELECT * INTO HRDB.dbo.EmployeesBackup FROM Employees

SELECT * FROM HRDB.dbo.EmployeesBackup
DROP TABLE HRDB.dbo.EmployeesBackup
```

```
--Copy only selected columns into a new table
SELECT * FROM Employees

SELECT Id, Name, Gender INTO EmployeesBackup FROM Employees

SELECT * FROM EmployeesBackup
DROP TABLE EmployeesBackup

--Copy only selected rows into a new table
SELECT * FROM Employees

SELECT * INTO EmployeesBackup FROM Employees WHERE DeptId = 1

SELECT * FROM EmployeesBackup
DROP TABLE EmployeesBackup

--Copy columns from 2 or more table into a new table
SELECt * FROm Departments
SELECT * FROM Employees
```

```
--Copy columns from 2 or more table into a new table
SELECt * FROm Departments
SELECT * FROM Employees

SELECT * INTO EmployeesBackup
FROM Employees
INNER JOIN Departments
ON Employees.DeptId = Departments.DepartmentId

SELECT * FROM EmployeesBackup
DROP TABLE EmployeesBackup
```

```
SELECT Employees.*, Departments.DepartmentName INTO EmployeesBackup
FROM Employees
INNER JOIN Departments
ON Employees.DeptId = Departments.DepartmentId

SELECT * FROM EmployeesBackup
DROP TABLE EmployeesBackup

--Create a new table whose columns and datatypes match with an existing
SELECT * INTO EmployeesBackup FROM Employees WHERE 1 <> 1
```

Results | Messages

Id	Name	Gender	Salary	DeptId	DepartmentName
1	Mark	Male	50000	1	IT
2	Sara	Female	65000	2	HR
3	Mike	Male	48000	3	Payroll
4	Pam	Female	70000	1	IT
5	John	Male	55000	2	HR

```
--You cannot use SELECT INTO statement to select data into an existing table
SELECT * INTO EmployeesBackup FROM Employees

-- Use INSERT INTO statement to select data into an existing table
INSERT INTO EmployeesBackup (Id, Name, Gender)
SELECT Id, Name, Gender FROM Employees


SELECT * FROM EmployeesBackup
```

Results | Messages

Id	Name	Gender	Salary	DeptId
1	Mark	Male	NULL	NULL
2	Sara	Female	NULL	NULL
3	Mike	Male	NULL	NULL
4	Pam	Female	NULL	NULL
5	John	Male	NULL	NULL

TO CREATE A TABLE ONLY WITH THE STRUCTURE OF THE EMPLOYEES TABLE AND NOT HAVE ANY ROWS COPIED IN TO THE NEW BACKUP TABLE USE THE FOLLOWING QUERY:

```
--Create a new table whose columns and datatypes match with an existing table.
SELECT * INTO EmployeesBackup FROM Employees WHERE 1 <> 1
                                   table SampleDB.dbo.Employees

SELECT * FROM EmployeesBackup
```

Difference between where and having

Difference between WHERE and HAVING

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Sales

Product	SaleAmount
iPhone	500
Laptop	800
iPhone	1000
Speakers	400
Laptop	600

Calculate total sales by product

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
```

Product	TotalSales
iPhone	1500
Laptop	1400
Speakers	400

Difference between WHERE and HAVING

WHERE clause filters rows before aggregate calculations are performed where as HAVING clause filters rows after aggregate calculations are performed

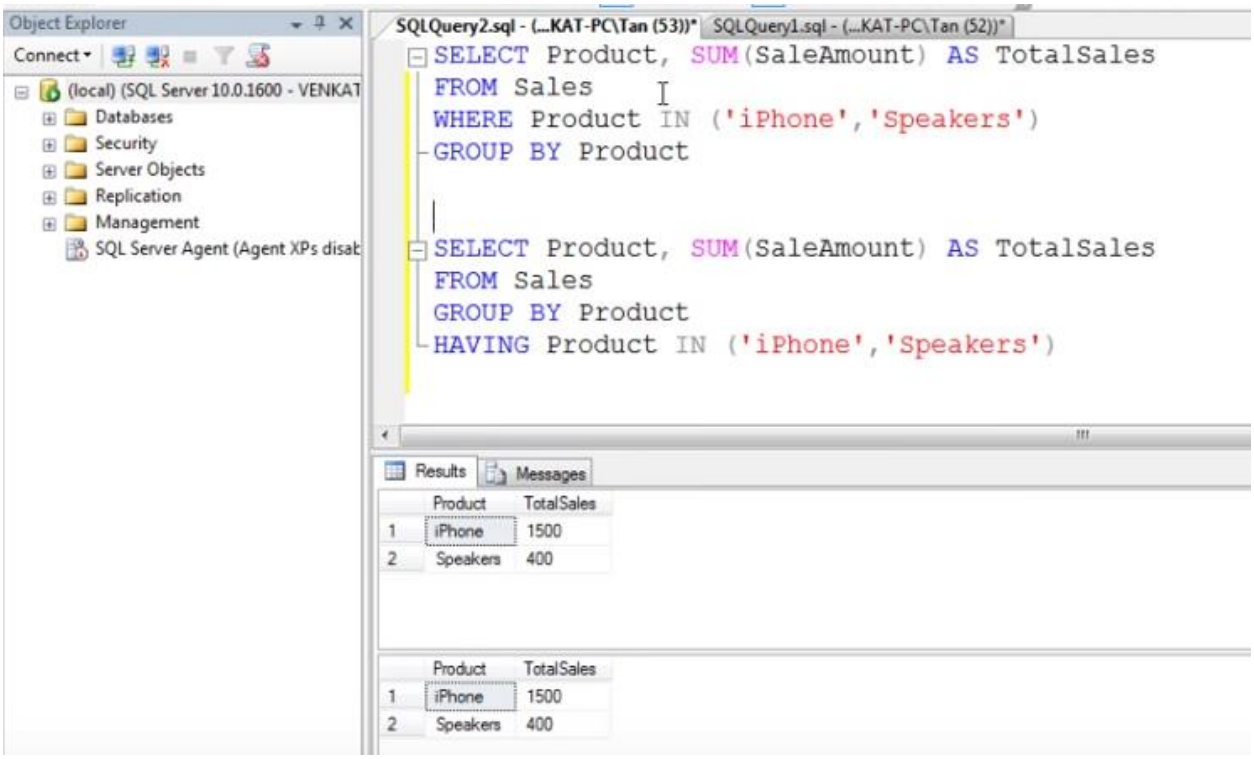
Total sales of iPhone and Speakers can be calculated by using either WHERE or HAVING clause

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product in ('iPhone', 'Speakers')
GROUP BY Product
```

Product	TotalSales
iPhone	1500
Speakers	400

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING Product in ('iPhone', 'Speakers')
```

WHERE clause filters rows before aggregate calculations are performed where as HAVING clause filters rows after aggregate calculations are performed. So from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.



Difference between WHERE and HAVING

WHERE clause cannot be used with aggregates where as HAVING can. This means WHERE clause is used for filtering individual rows where as HAVING clause is used to filter groups.

WHERE comes before GROUP BY. This means WHERE clause filters rows before aggregate calculations are performed. HAVING comes after GROUP BY. This means HAVING clause filters rows after aggregate calculations are performed. So from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.

WHERE and HAVING can be used together in a SELECT query. In this case WHERE clause is applied first to filter individual rows. The rows are then grouped and aggregate calculations are performed, and then the HAVING clause filter the groups.

Table valued parameters

Table Valued Parameters in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Table Valued Parameter is a new feature introduced in SQL SERVER 2008. Table Valued Parameter allows a table (i.e multiple rows of data) to be passed as a parameter to a stored procedure from T-SQL code or from an application. Prior to SQL SERVER 2008, it is not possible to pass a table variable to a stored procedure.

Please note : Table valued parameters must be passed as read-only to stored procedures, functions etc. This means you cannot perform DML operations like INSERT, UPDATE or DELETE on a table-valued parameter in the body of a function, stored procedure etc.

3 steps to pass multiple rows to a stored procedure using Table Valued Parameter
Step 1 : Create User-defined Table Type
Step 2 : Use the User-defined Table Type as a parameter in the stored procedure.
Step 3 : Declare a table variable, insert the data rows and then pass the table variable as a parameter to the stored procedure

Step 1:

```
Create Type EmpTableType as Table
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10)
)
```

Messages

```
Command(s) completed successfully.
```

Step 2:

```
Create Proc spInsertEmployees
@EmpTableType EmpTableType READONLY
AS
BEGIN
    Insert Into Employees
    Select * from @EmpTableType
END
```

Step 3:

```
Declare @EmployeeTableType EmpTableType

Insert into @EmployeeTableType values (1, 'Mark', 'Male')
INSERT INTO @EmployeeTableType VALUES (2, 'Mary', 'Female')
INSERT INTO @EmployeeTableType VALUES (3, 'John', 'Male')
INSERT INTO @EmployeeTableType VALUES (4, 'Sara', 'Female')
INSERT INTO @EmployeeTableType VALUES (5, 'Rob', 'Male')

Execute spInsertEmployees @EmployeeTableType
```

Check:

```
Select * from Employees
```

Results | Messages

	Id	Name	Gender
1	1	Mark	Male
2	2	Mary	Female
3	3	John	Male
4	4	Sara	Female
5	5	Rob	Male

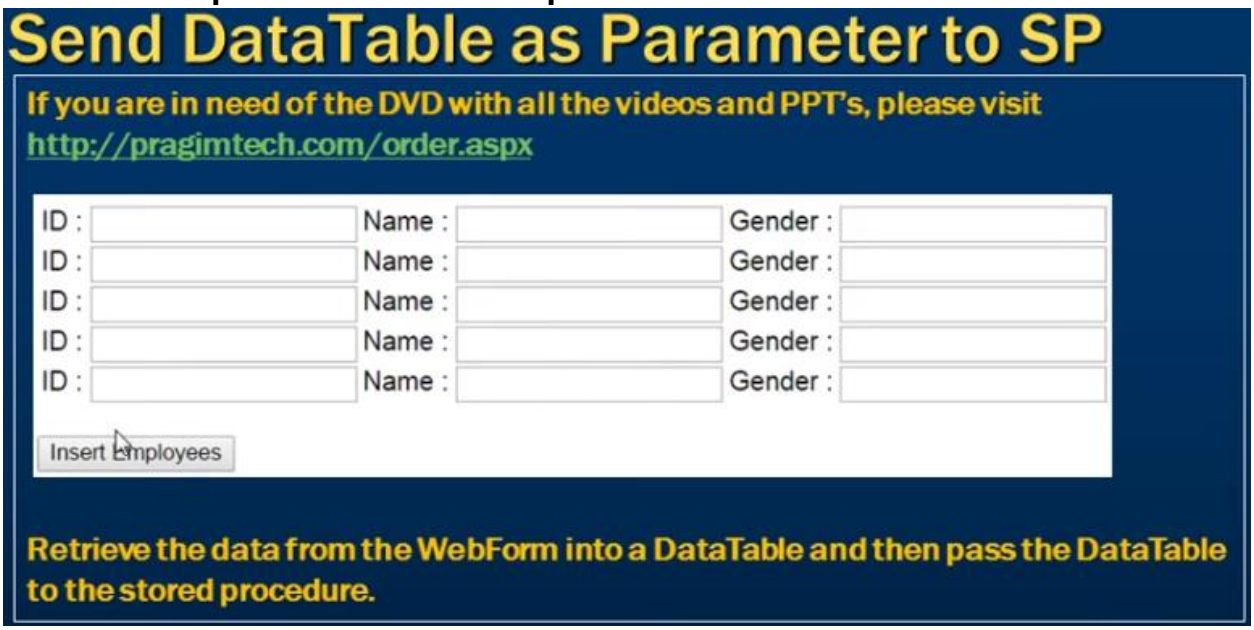
Table Valued Parameters in SQL Server

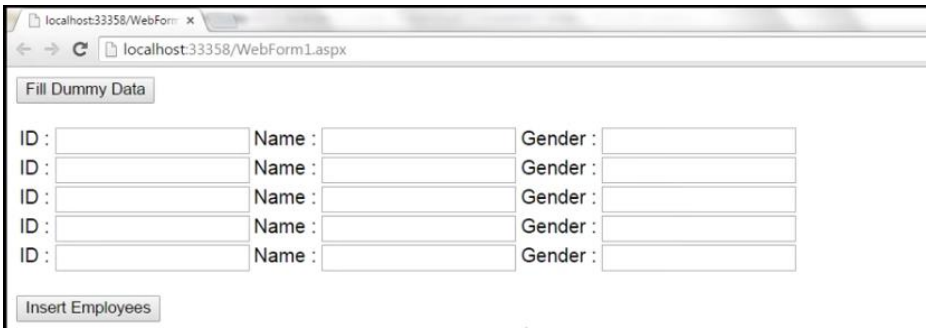
Step 3 : Declare a table variable, insert the data and then pass the table variable as a parameter to the stored procedure

```
-- Declare the Table Variable
DECLARE @EmployeeTableType EmpTableType

-- Insert rows into the Table variable that you want to send to the SP
INSERT INTO @EmployeeTableType VALUES (1, 'Mark', 'Male')
INSERT INTO @EmployeeTableType VALUES (2, 'Mary', 'Female')
INSERT INTO @EmployeeTableType VALUES (3, 'John', 'Male')
INSERT INTO @EmployeeTableType VALUES (4, 'Sara', 'Female')
INSERT INTO @EmployeeTableType VALUES (5, 'Rob', 'Male')

-- Pass the table variable as a parameter to the stored procedure
EXECUTE spInsertEmployees @EmployeeTableType
```

Next Video : How to pass table as a parameter to the stored procedure from an ADO.NET application

Send datatable as parameter to stored procedure



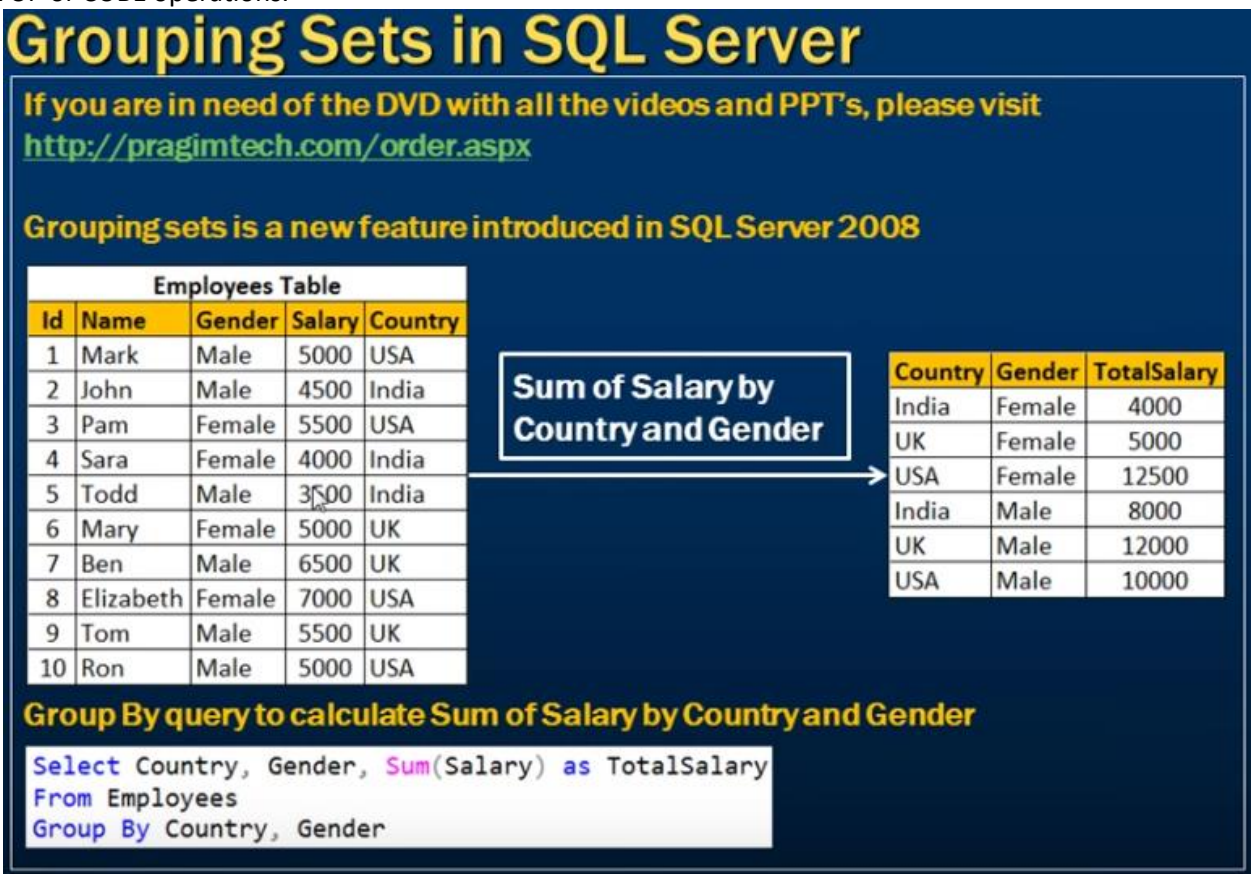



```
SQLQuery7.sql - V...KAT-PC\Tan (54))*
Select * from Employees

Create Proc spInsertEmployees
@EmpTableType EmpTableType READONLY
AS
BEGIN
    Insert Into Employees
    Select * from @EmpTableType
END
```

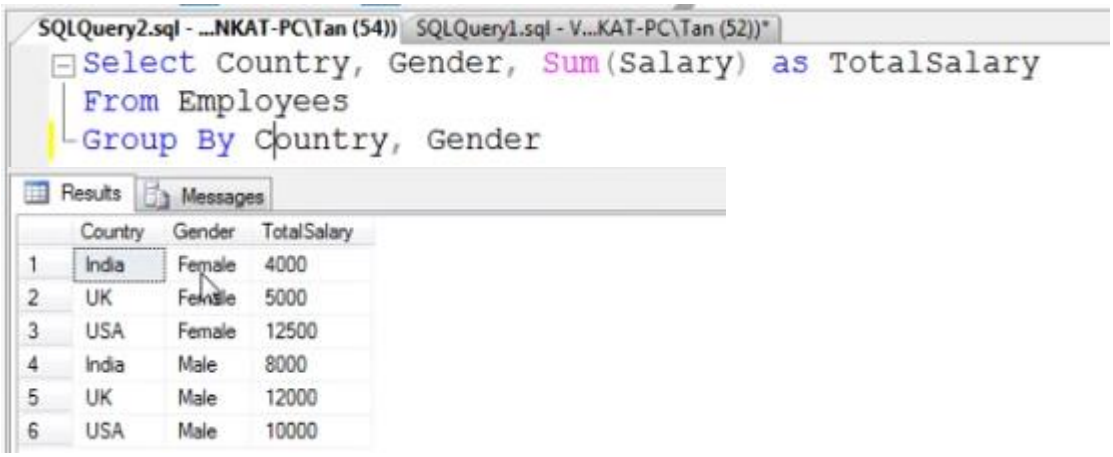
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Demo.WebForm

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body style="font-family: Arial">
    <form id="form1" runat="server">
        <asp:Button ID="btnFillDummyData" runat="server" Text="Fill Dummy Data" />
        <br />
        <br />
        <table>...</table>
        <br />
        <asp:Button ID="btnInsert" runat="server" Text="Insert Employees"/>
    </form>
</body>
</html>
```

Grouping Sets

GROUPING SET is able to generate a result **set** that can be generated by a UNION ALL of multiple simple GROUP BY clauses. It is capable of generating a result **set** that is equivalent to the result **set** generated by ROLL UP or CUBE operations.



We need to write a query to sum the salary by country column first then by gender. The result is shown in the right table. For this we can write a simple Group By query as shown.



Along with this we also want the sum of salary grouped just by country. For this purpose we use another Group By clause as shown below and apply Union All operator between two Group By queries.

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500

Sum of Salary by Country

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country
```

Let's say we also want salaries grouped by gender.

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country, and also Sum of Salary just by Gender

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500
NULL	Female	21500
NULL	Male	30000

Sum of Salary by Country

Sum of Salary by Gender

Hence, we write another Group By query very similar to above queries. Moreover, we also want Grand Total.

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country, and also Sum of Salary just by Gender and finally grand total

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500
NULL	Female	21500
NULL	Male	30000
NULL	NULL	51500

Sum of Salary by Country

Sum of Salary by Gender

Grand Total

SQLQuery2.sql - V...KAT-PC\Tan (54))* | SQLQuery1.sql - V...KAT-PC\Tan (52))*

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country

UNION ALL

Select NULL, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Gender

UNION ALL

Select NULL, NULL, Sum(Salary) as TotalSalary
From Employees
```

When we execute this query we get the output as shown in the slide above.

However, this query is huge, also this query is going to hit Employees table 4 times. So this not very good from performance standpoint.

If we use Grouping Set feature then code size will be greatly reduced.

TO ACHIEVE THE SAME RESULT AS IN UNION ALL, WE CAN USE GROUPING SET

Grouping Sets in SQL Server

With Grouping Sets feature introduced in SQL Server 2008, the amount of T-SQL code that you have to write will be greatly reduced

```
Select Country, Gender, Sum(Salary) TotalSalary
From Employees
Group BY
 GROUPING SETS
 (
   (Country, Gender), -- Sum of Salary by Country & Gender
   (Country),         -- Sum of Salary by Country
   (Gender) ,         -- Sum of Salary by Gender
   ()                 -- Grand Total
 )
```

SQLQuery5.sql - V...KAT-PC\Tan (53))* | SQLQuery2.sql - V...KAT-PC\Tan (54))* | SQLQuery1.sql - V...KAT-PC\Tan (52))*

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By
    GROUPING SETS
    (
        (Country, Gender),
        (Country),
        (Gender),
        ()
    )
Order By GROUPING(Country), GROUPING(Gender), Gender
```

This gives the same result.

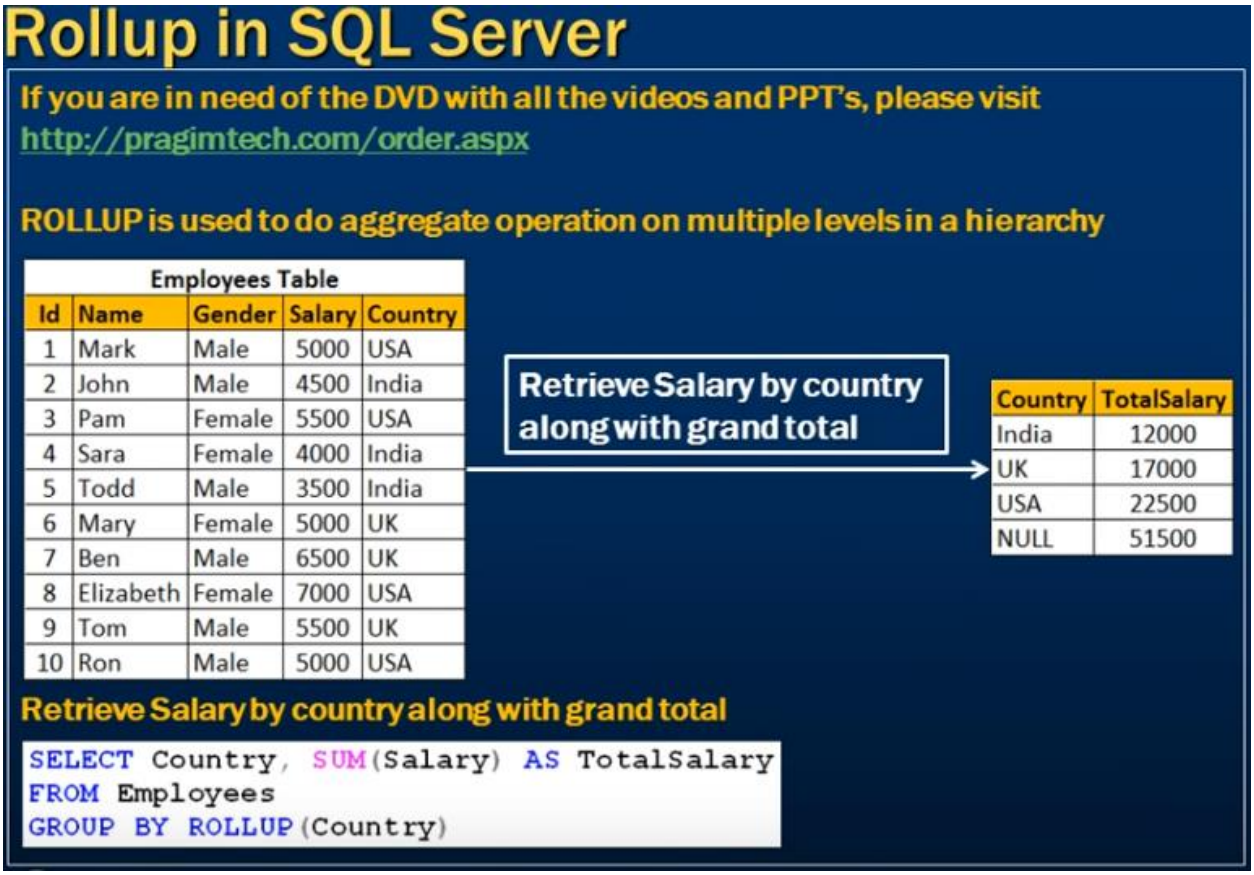
Results | Messages

	Country	Gender	TotalSalary
1	India	Female	4000
2	UK	Female	5000
3	USA	Female	12500
4	India	Male	8000
5	UK	Male	12000
6	USA	Male	10000
7	India	NULL	12000
8	UK	NULL	17000
9	USA	NULL	22500
10	NULL	Female	21500
11	NULL	Male	30000
12	NULL	NULL	51500

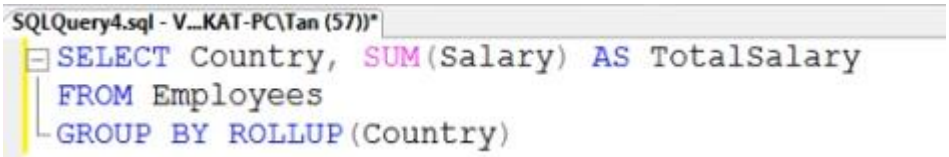
Rollup

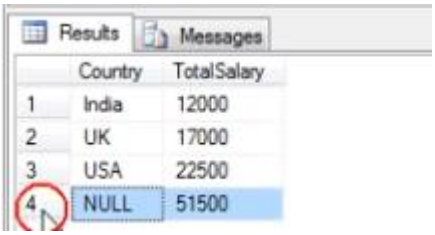
The **ROLLUP** is an extension of the GROUP BY clause. The **ROLLUP** option allows you to include extra rows that represent the subtotals, which are commonly referred to as super-aggregate rows, along with the grand total row. By using the **ROLLUP** option, you can use a single query to generate multiple grouping sets.

ROLL UP ALSO RETURNS THE AGGREGATE OF ALL THE ROWS WITH NULL IN THE LAST RECORD ALONG WITH GROUP BY



We want to retrieve salary by country along with grand total.





Rollup queries can be written in two different ways giving same result.

SQLQuery4.sql - V...KAT-PC\Tan (57))*

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY ROLLUP(Country)


SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country with ROLLUP
```

Results | Messages

	Country	TotalSalary
1	India	12000
2	UK	17000
3	USA	22500
4	NULL	51500

	Country	TotalSalary
1	India	12000
2	UK	17000
3	USA	22500
4	NULL	51500

Rollup in SQL Server

Other ways to retrieve Salary by Country along with Grand Total

```
-- Using ROLLUP clause with GROUP BY
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country WITH ROLLUP
```

```
-- Using UNION ALL operator along with GROUP BY
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

```
--Using GROUPING SETS
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
    (Country),
    ()
)
```

We can use Rollup to get Group Salary by Country, Gender, and also compute Subtotal at Country level and Grand Total.

Rollup in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Grouping sets is a new feature introduced in SQL Server 2008

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Group Salary by Country and Gender. Also compute the Subtotal at Country level and Grand Total.

Country	Gender	TotalSalary
India	Female	4000
India	Male	8000
India	NULL	12000
UK	Female	5000
UK	Male	12000
UK	NULL	17000
USA	Female	12500
USA	Male	10000
USA	NULL	22500
NULL	NULL	51500

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY ROLLUP(Country, Gender)
```

Rollup in SQL Server

Other ways to group Salary by Country and Gender and also to compute the Subtotal at Country level and Grand Total

```
--Using GROUPING SETS
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
    (Country, Gender),
    (Country),
    ()
)
```

Cube Function

Similar to the ROLLUP , **CUBE** is an extension of the GROUP BY clause. **CUBE** allows you to generate subtotals like the ROLLUP extension. In addition, the **CUBE** extension will generate subtotals for all combinations of grouping columns specified in the GROUP BY clause.

CUBE FUNCTION SAME AS GROUPING SETS:

Cube in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Cube() in SQL Server produces the result set by generating all combinations of columns specified in GROUP BY CUBE()

Write a query to retrieve Sum of Salary grouped by all combinations of the 2 columns (Country & Gender) as well as Grand Total.

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
NULL	Female	21500
India	Male	8000
UK	Male	12000
USA	Male	10000
NULL	Male	30000
NULL	NULL	51500
India	NULL	12000
UK	NULL	17000
USA	NULL	22500

SQLQuery6.sql - V...KAT-PC\Tan (54))*

```
Select Country, Gender, SUM(Salary) as TotalSalary
from Employees
Group By Cube(Country, Gender)
```

Results | Messages

	Country	Gender	TotalSalary
1	India	Female	4000
2	UK	Female	5000
3	USA	Female	12500
4	NULL	Female	21500
5	India	Male	8000
6	UK	Male	12000
7	USA	Male	10000
8	NULL	Male	30000
9	NULL	NULL	51500
10	India	NULL	12000
11	UK	NULL	17000
12	USA	NULL	22500

```
SQLQuery6.sql - V...KAT-PC\Tan (54))*
Select Country, Gender, SUM(Salary) as TotalSalary
from Employees
Group By Cube(Country, Gender)

Select Country, Gender, SUM(Salary) as TotalSalary
from Employees
Group By Country, Gender with Cube

Select Country, Gender, SUM(Salary) as TotalSalary
from Employees
Group By
    GROUPING SETS
    (
        (Country, Gender),
        Country,
        Gender,
        ()
    )
```

```
Select Country, Gender, SUM(Salary) as TotalSalary
from Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, SUM(Salary) as TotalSalary
from Employees
Group By Country

UNION ALL

Select NULL, Gender, SUM(Salary) as TotalSalary
from Employees
Group By Gender

UNION ALL

Select NULL, NULL, SUM(Salary) as TotalSalary
from Employees
```

Difference between cube and rollup

Difference between CUBE and ROLLUP

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

CUBE generates a result set that shows aggregates for all combinations of values in the selected columns, where as ROLLUP generates a result set that shows aggregates for a hierarchy of values in the selected columns

Continent	Country	City	SaleAmount
Asia	India	Bangalore	1000
Asia	India	Chennai	2000
Asia	Japan	Tokyo	4000
Asia	Japan	Hiroshima	5000
Europe	United Kingdom	London	1000
Europe	United Kingdom	Manchester	2000
Europe	France	Paris	4000
Europe	France	Cannes	5000

ROLLUP(Continent, Country, City)

Continent, Country, City
Continent, Country,
Continent
()

CUBE(Continent, Country, City)

Continent, Country, City
Continent, Country,
Continent, City
Continent
Country, City
Country,
City
()

SQLQuery2.sql - V...KAT-PC\Tan (55))* SQLQuery1.sql - V...KAT-PC\Tan (54))*

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales
1	Asia	India	Bangalore	1000
2	Asia	India	Chennai	2000
3	Asia	India	NULL	3000
4	Asia	Japan	Hiroshima	5000
5	Asia	Japan	Tokyo	4000
6	Asia	Japan	NULL	9000
7	Asia	NULL	NULL	12000
8	Europe	France	Cannes	5000
9	Europe	France	Paris	4000
10	Europe	France	NULL	9000
11	Europe	United Kingdom	London	1000
12	Europe	United Kingdom	Manchester	2000
13	Europe	United Kingdom	NULL	3000
14	Europe	NULL	NULL	12000
15	NULL	NULL	NULL	24000

SQLQuery2.sql - ...NKAT-PC\Tan (55)) | SQLQuery1.sql - V...KAT-PC\Tan (54))*

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY CUBE(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales
1	Asia	India	Bangalore	1000
2	NULL	India	Bangalore	1000
3	NULL	NULL	Bangalore	1000
4	Europe	France	Cannes	5000
5	NULL	France	Cannes	5000
6	NULL	NULL	Cannes	5000
7	Asia	India	Chennai	2000
8	NULL	India	Chennai	2000
9	NULL	NULL	Chennai	2000
10	Asia	Japan	Hiroshima	5000
11	NULL	Japan	Hiroshima	5000
12	NULL	NULL	Hiroshima	5000
13	Europe	United Kingdom	London	1000
14	NULL	United Kingdom	London	1000
15	NULL	NULL	London	1000
16	Europe	United Kingdom	Manchester	2000

Query executed successfully.

Results | Messages

```
(43 row(s) affected)
```

Difference between CUBE and ROLLUP

You won't see any difference when you use ROLLUP and CUBE on a single column

```
SELECT Continent, Sum(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent)

-- OR

SELECT Continent, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY CUBE(Continent)
```

Continent	TotalSales
Asia	12000
Europe	12000
NULL	24000

Grouping function

SQL GROUPING Function. The **SQL GROUPING** is one of the **SQL** Aggregate **Function**, which is used to indicate whether the specified column in a **GROUP** BY Clause aggregated or not. This **Grouping function** will return one for aggregated and zero for not aggregated.

Grouping function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is Grouping function

Grouping(Column) indicates whether the column in a GROUP BY list is aggregated or not. Grouping returns 1 for aggregated or 0 for not aggregated in the result set.

Continent	Country	City	TotalSales	GP_Continent	GP_Country	GP_City
Asia	India	Bangalore	1000	0	0	0
Asia	India	Chennai	2000	0	0	0
Asia	India	NULL	3000	0	0	1
Asia	Japan	Hiroshima	5000	0	0	0
Asia	Japan	Tokyo	4000	0	0	0
Asia	Japan	NULL	9000	0	0	1
Asia	NULL	NULL	12000	0	1	1
Europe	France	Cannes	5000	0	0	0
Europe	France	Paris	4000	0	0	0
Europe	France	NULL	9000	0	0	1
Europe	United Kingdom	London	1000	0	0	0
Europe	United Kingdom	Manchester	2000	0	0	0
Europe	United Kingdom	NULL	3000	0	0	1
Europe	NULL	NULL	12000	0	1	1
NULL	NULL	NULL	24000	1	1	1

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        GROUPING(Continent) as GP_Continent,
        GROUPING(Country) as GP_Country,
        GROUPING(City) as GP_City
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

Continent	Country	City	TotalSales	GP_Continent	GP_Country	GP_City
Asia	India	Bangalore	1000	0	0	0
Asia	India	Chennai	2000	0	0	0
Asia	India	NULL	3000	0	0	1
Asia	Japan	Hiroshima	5000	0	0	0
Asia	Japan	Tokyo	4000	0	0	0
Asia	Japan	NULL	9000	0	0	1
Asia	NULL	NULL	12000	0	1	1
Europe	France	Cannes	5000	0	0	0
Europe	France	Paris	4000	0	0	0
Europe	France	NULL	9000	0	0	1
Europe	United Kingdom	London	1000	0	0	0
Europe	United Kingdom	Manchester	2000	0	0	0
Europe	United Kingdom	NULL	3000	0	0	1
Europe	NULL	NULL	12000	0	1	1
NULL	NULL	NULL	24000	1	1	1

SQLQuery4.sql - V...KAT-PC\Tan (54))*

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)

--Select * from Sales
--Update Sales Set City = NULL where Id = 1
```

Results | Messages

	Continent	Country	City	TotalSales
1	Asia	India	Bangalore	1000
2	Asia	India	Chennai	2000
3	Asia	India	NULL	3000
4	Asia	Japan	Hiroshima	5000
5	Asia	Japan	Tokyo	4000
6	Asia	Japan	NULL	9000
7	Asia	NULL	NULL	12000
8	Europe	France	Cannes	5000
9	Europe	France	Paris	4000
10	Europe	France	NULL	9000
11	Europe	United Kingdom	London	1000
12	Europe	United Kingdom	Manchester	2000
13	Europe	United Kingdom	NULL	3000
14	Europe	NULL	NULL	12000
15	NULL	NULL	NULL	24000

SQLQuery4.sql - V...KAT-PC\Tan (54))*

```
SELECT
Case When GROUPING(Continent) = 1 THEN 'All' ELSE ISNULL(Continent, 'Unknown') END AS Cont
Case When GROUPING(Country) = 1 THEN 'All' ELSE ISNULL(Country, 'Unknown') END AS Country,
Case When GROUPING(City) = 1 THEN 'All' ELSE ISNULL(City, 'Unknown') END AS City,
SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales
1	Asia	India	Bangalore	1000
2	Asia	India	Chennai	2000
3	Asia	India	All	3000
4	Asia	Japan	Hiroshima	5000
5	Asia	Japan	Tokyo	4000
6	Asia	Japan	All	9000
7	Asia	All	All	12000
8	Europe	France	Cannes	5000
9	Europe	France	Paris	4000
10	Europe	France	All	9000
11	Europe	United Kingdom	London	1000
12	Europe	United Kingdom	Manchester	2000
13	Europe	United Kingdom	All	3000
14	Europe	All	All	12000
15	All	All	All	24000

Grouping function in SQL Server

Can't I use ISNULL function instead of GROUPING function to achieve the same

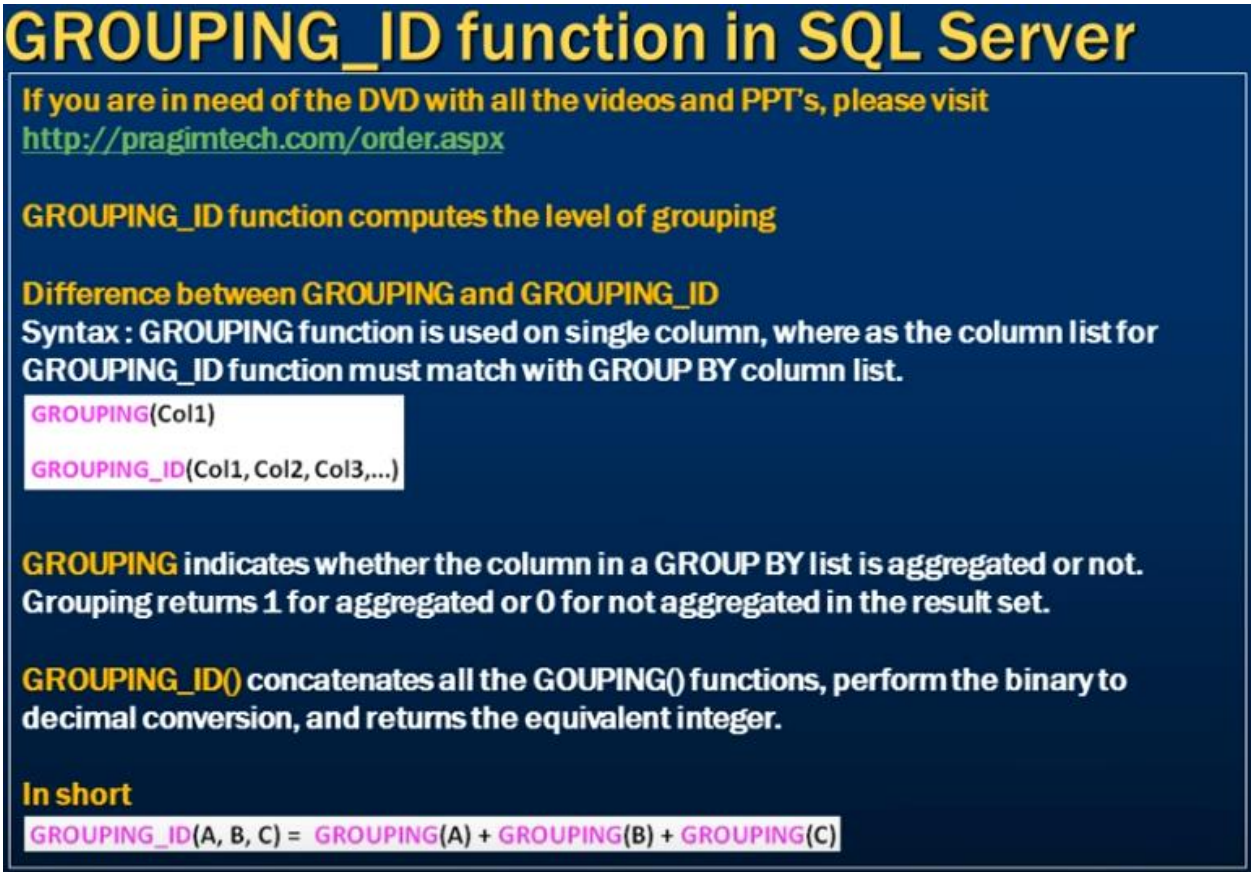
You can, but only if your data does not contain NULL values

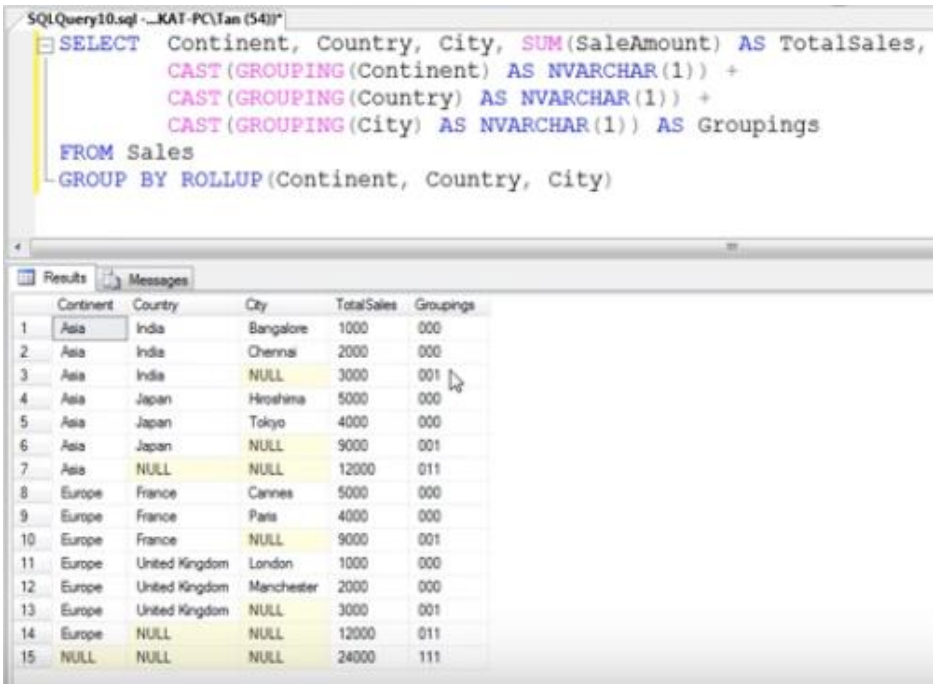
```
SELECT   ISNULL(Continent, 'All') AS Continent,
         ISNULL(Country, 'All') AS Country,
         ISNULL(City, 'All') AS City,
         SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Please note : Grouping function can be used with Rollup, Cube and Grouping Sets

GROUPING ID function

The **SQL GROUPING_ID** is one of the **SQL** Aggregate **Functions**, which is used to compute the level of Grouping. This **GROUPING_ID function** will return the integer value.





```
SELECT  Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        CAST(GROUPING(Continent) AS NVARCHAR(1)) +
        CAST(GROUPING(Country) AS NVARCHAR(1)) +
        CAST(GROUPING(City) AS NVARCHAR(1)) AS Groupings
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales	Groupings
1	Asia	India	Bangalore	1000	000
2	Asia	India	Chennai	2000	000
3	Asia	India	NULL	3000	001
4	Asia	Japan	Hiroshima	5000	000
5	Asia	Japan	Tokyo	4000	000
6	Asia	Japan	NULL	9000	001
7	Asia	NULL	NULL	12000	011
8	Europe	France	Cannes	5000	000
9	Europe	France	Paris	4000	000
10	Europe	France	NULL	9000	001
11	Europe	United Kingdom	London	1000	000
12	Europe	United Kingdom	Manchester	2000	000
13	Europe	United Kingdom	NULL	3000	001
14	Europe	NULL	NULL	12000	011
15	NULL	NULL	NULL	24000	111

GROUPING_ID() concatenates all the GOUPING() functions, perform the binary to decimal conversion, and returns the equivalent integer.

In short

GROUPING_ID(A, B, C) = GROUPING(A) + GROUPING(B) + GROUPING(C)

SQLQuery10.sql -...KAT-PC\Tan (54))*

```
SELECT  Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        CAST(GROUPING(Continent) AS NVARCHAR(1)) +
        CAST(GROUPING(Country) AS NVARCHAR(1)) +
        CAST(GROUPING(City) AS NVARCHAR(1)) AS Groupings,
        GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales	Groupings	GPID
1	Asia	India	Bangalore	1000	000	0
2	Asia	India	Chennai	2000	000	0
3	Asia	India	NULL	3000	001	1
4	Asia	Japan	Hiroshima	5000	000	0
5	Asia	Japan	Tokyo	4000	000	0
6	Asia	Japan	NULL	9000	001	1
7	Asia	NULL	NULL	12000	011	3
8	Europe	France	Cannes	5000	000	0
9	Europe	France	Paris	4000	000	0
10	Europe	France	NULL	9000	001	1
11	Europe	United Kingdom	London	1000	000	0
12	Europe	United Kingdom	Manchester	2000	000	0
13	Europe	United Kingdom	NULL	3000	001	1
14	Europe	NULL	NULL	12000	011	3
15	NULL	NULL	NULL	24000	111	7

DECIMAL TO BINARY CONVERSION:

1	1	1	
2 Power 2 X 1	2 Power 1 X 1	2 Power 0 X 1	
4	2	1	7

SQLQuery10.sql -...KAT-PC\Tan (54))*

```
SELECT  Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        CAST(GROUPING(Continent) AS NVARCHAR(1)) +
        CAST(GROUPING(Country) AS NVARCHAR(1)) +
        CAST(GROUPING(City) AS NVARCHAR(1)) AS Groupings,
        GROUPING_ID(Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Results | Messages

	Continent	Country	City	TotalSales	Groupings	GPID
1	Asia	India	Bangalore	1000	000	0
2	Asia	India	Chennai	2000	000	0
3	Asia	India	NULL	3000	001	1
4	Asia	Japan	Hiroshima	5000	000	0
5	Asia	Japan	Tokyo	4000	000	0
6	Asia	Japan	NULL	9000	001	1
7	Asia	NULL	NULL	12000	011	3
8	Europe	France	Cannes	5000	000	0
9	Europe	France	Paris	4000	000	0
10	Europe	France	NULL	9000	001	1
11	Europe	United Kingdom	London	1000	000	0
12	Europe	United Kingdom	Manchester	2000	000	0
13	Europe	United Kingdom	NULL	3000	001	1
14	Europe	NULL	NULL	12000	011	3
15	NULL	NULL	NULL	24000	111	3

Sorting by Level of Grouping

```
SELECT  Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
ORDER BY GPID
```

Continent	Country	City	TotalSales	GPID
Asia	Japan	Hiroshima	5000	0
Asia	Japan	Tokyo	4000	0
Asia	India	Bangalore	1000	0
Asia	India	Chennai	2000	0
Europe	France	Cannes	5000	0
Europe	France	Paris	4000	0
Europe	United Kingdom	London	1000	0
Europe	United Kingdom	Manchester	2000	0
Europe	United Kingdom	NULL	3000	1
Europe	France	NULL	9000	1
Asia	India	NULL	3000	1
Asia	Japan	NULL	9000	1
Asia	NULL	NULL	12000	3
Europe	NULL	NULL	12000	3
NULL	NULL	NULL	24000	7

FILTERING IS ALSO POSSIBLE WITH HAVING CLAUSE:

Filter by Level of Grouping

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
HAVING GROUPING_ID(Continent, Country, City) = 3
```

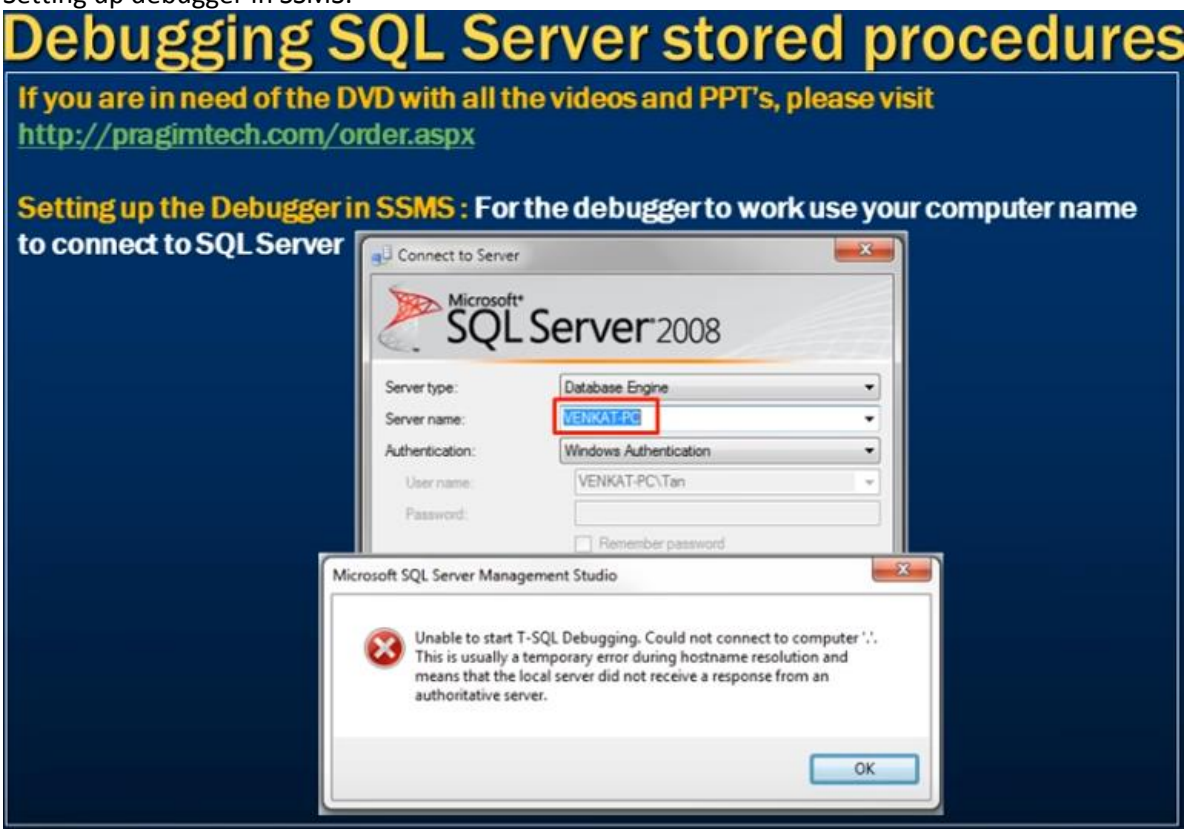
Retrieves only CONTINENT level aggregated data

Continent	Country	City	TotalSales	GPID
Asia	NULL	NULL	12000	3
Europe	NULL	NULL	12000	3

Debugging stored procedures

- How to debug stored procedures in SQL Server

Setting up debugger in SSMS.



We want to debug following code:

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
DECLARE @TargetNumber INT
SET @TargetNumber = 10
EXECUTE spPrintEvenNumbers @TargetNumber
Print 'Done'
```

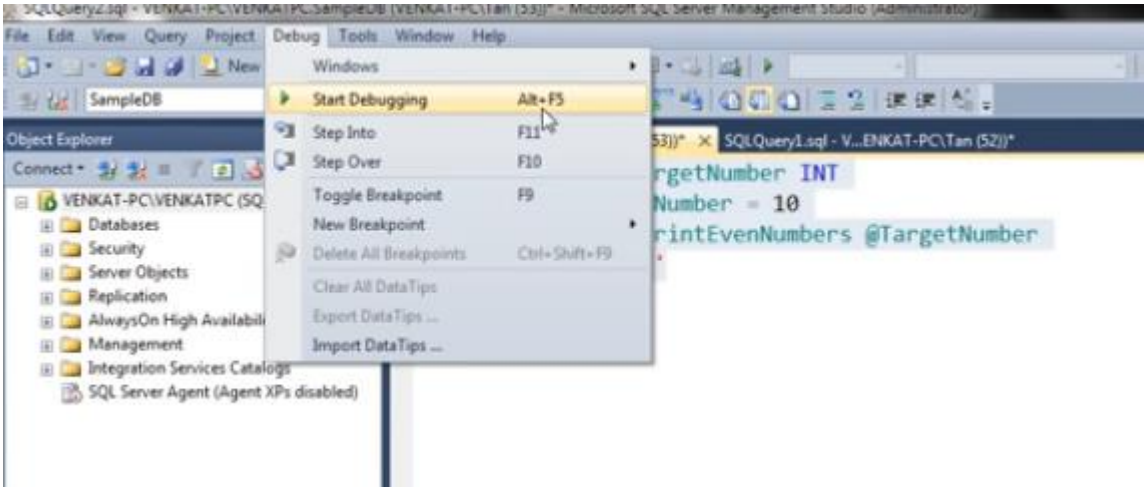
We are calling the following stored procedure in above code.

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
Create procedure spPrintEvenNumbers
@Target int
as
Begin
    Declare @StartNumber int
    Set @StartNumber = 1

    while(@StartNumber < @Target)
    Begin
        If(@StartNumber%2 = 0)
        Begin
            Print @StartNumber
        End
        Set @StartNumber = @StartNumber + 1
    End
    Print 'Finished printing even numbers till ' + RTRIM(@Target)
End
```

If we supply 10 as input parameter @Target to this stored procedure it prints all even numbers from 0 till 10. We want to debug this T-SQL code.

To start debugger select “Start Debugging” from “Debug” menu.



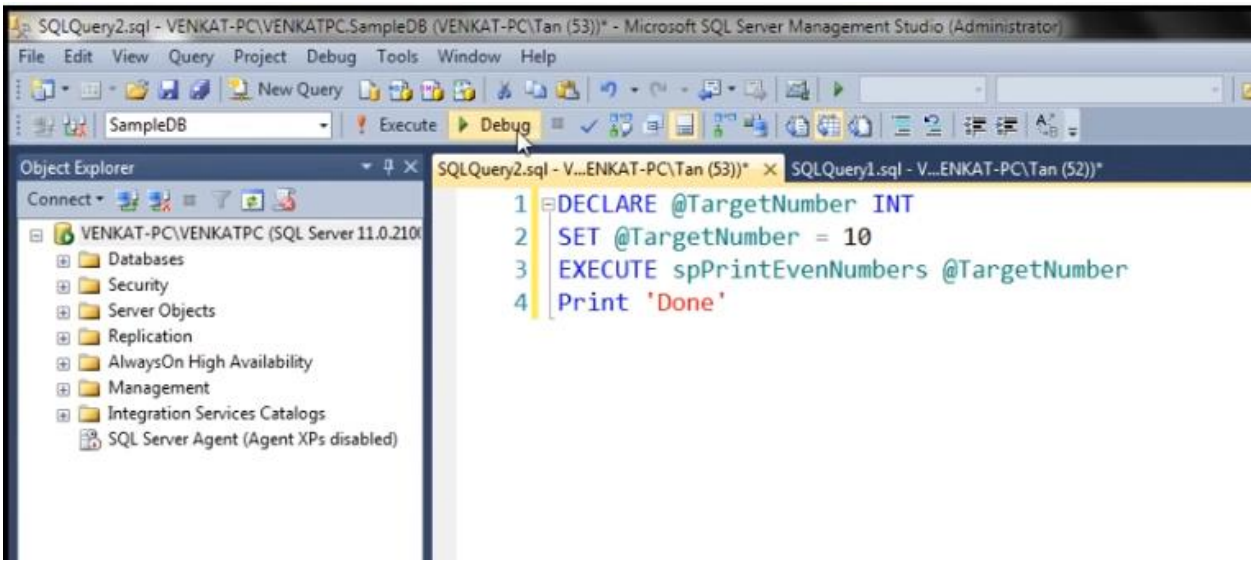
If there is error in starting debugger: the most likely reason can be that you used dot (.) or localhost as your server name, to fix this error all you have to do is use your computer name as Server name.

So, to start debugging you can use Alt+F5 also.

Debugging SQL Server stored procedures

Start Debugging	ALT + F5
Stepping thru the code	
Step Over	F10
Step Into	F11
Step Out	SHIFT + F11
Stop Debugging	SHIFT + F5
Show Next Statement	Shows the next statement that the debugger is about to execute
Run to Cursor	Executes all the statements in a batch up to the current cursor position
Locals Window	Displays the current values of variables and parameters
Watch Window	Watch the values of variables
Call Stack	Allows you to navigate up and down the call stack to see what values your application is storing at different levels. It's an invaluable tool for determining why your code is doing what it's doing.
Immediate Window	Very helpful during debugging to evaluate expressions, and print variable values

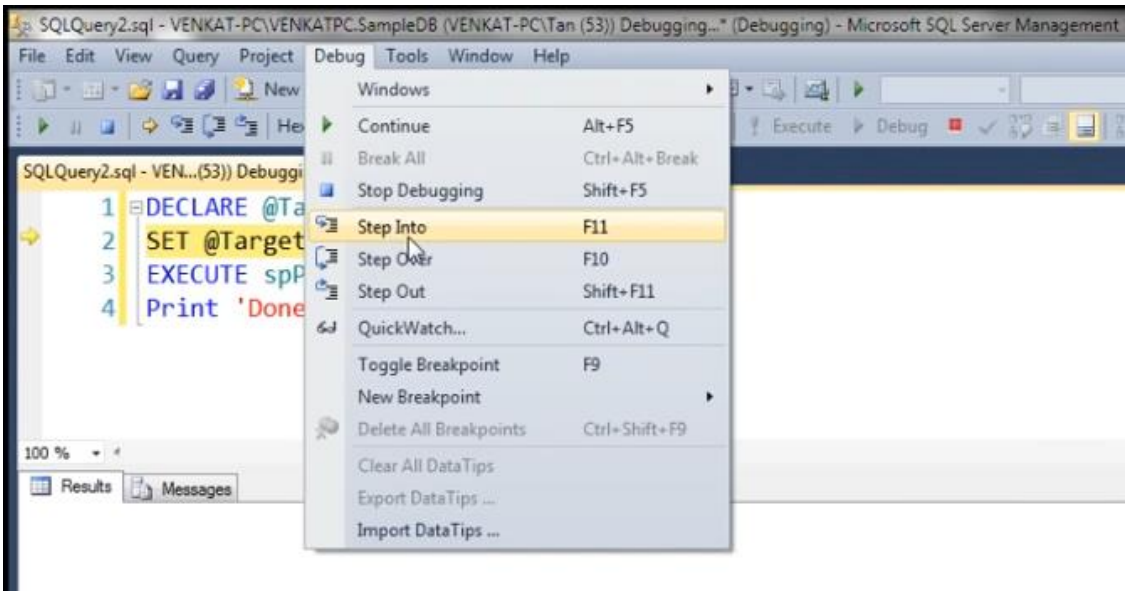
Or, simply click Debug button.



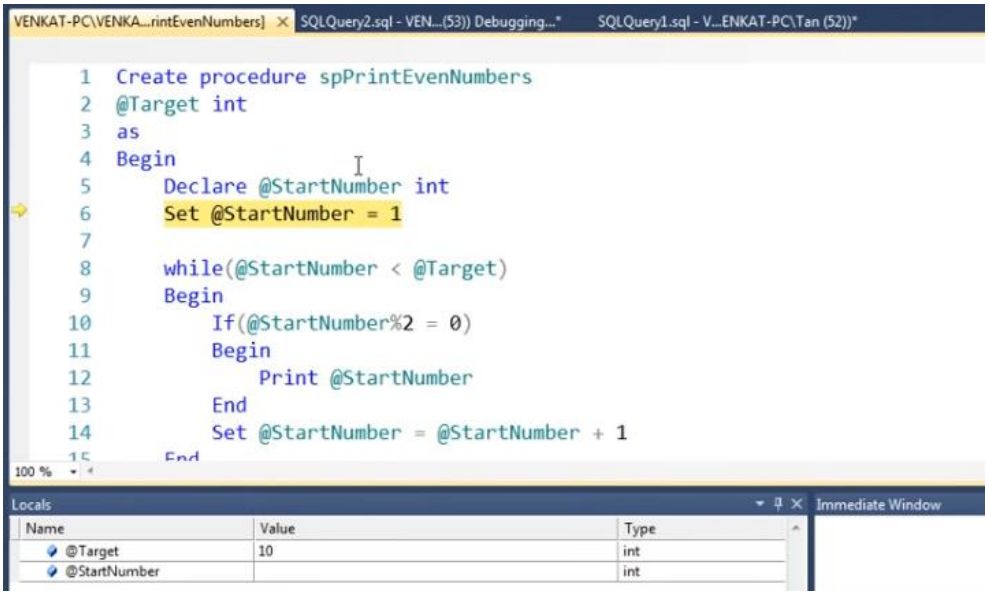
The line to be debugged gets highlighted in yellow.

```
DECLARE @TargetNumber INT
SET @TargetNumber = 10
EXECUTE spPrintEvenNumbers @TargetNumber
Print 'Done'
```

To debug we can use key shortcuts as given above or click menu as below:



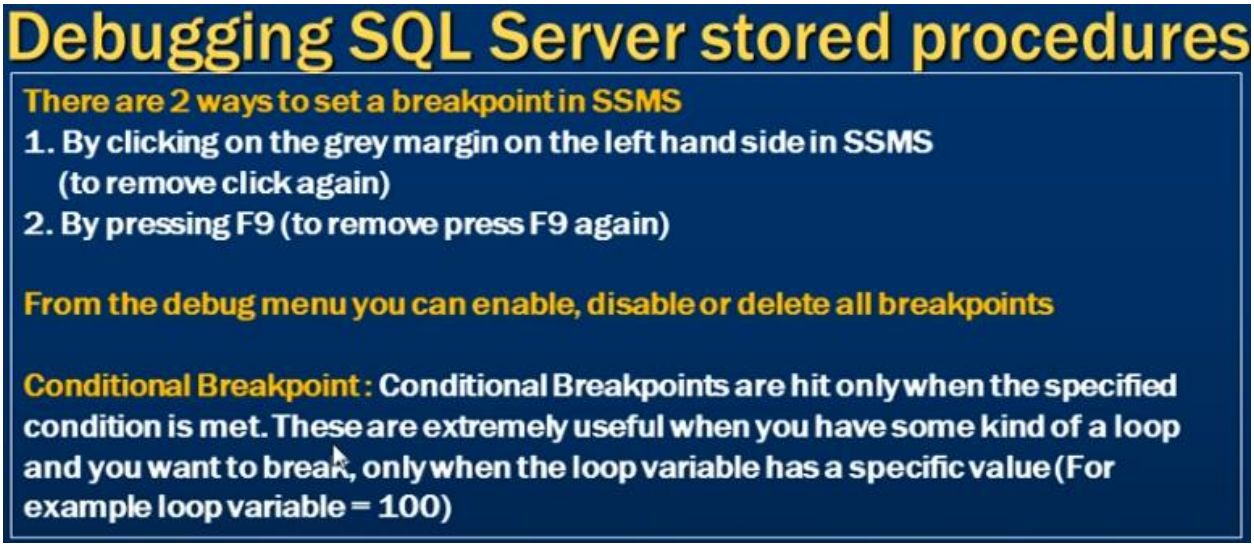
The line to be debugged gets marked with an arrow. If click Step Over then we do not get to view the code being debugged, but if we click Step Into then code being debugged is displayed.

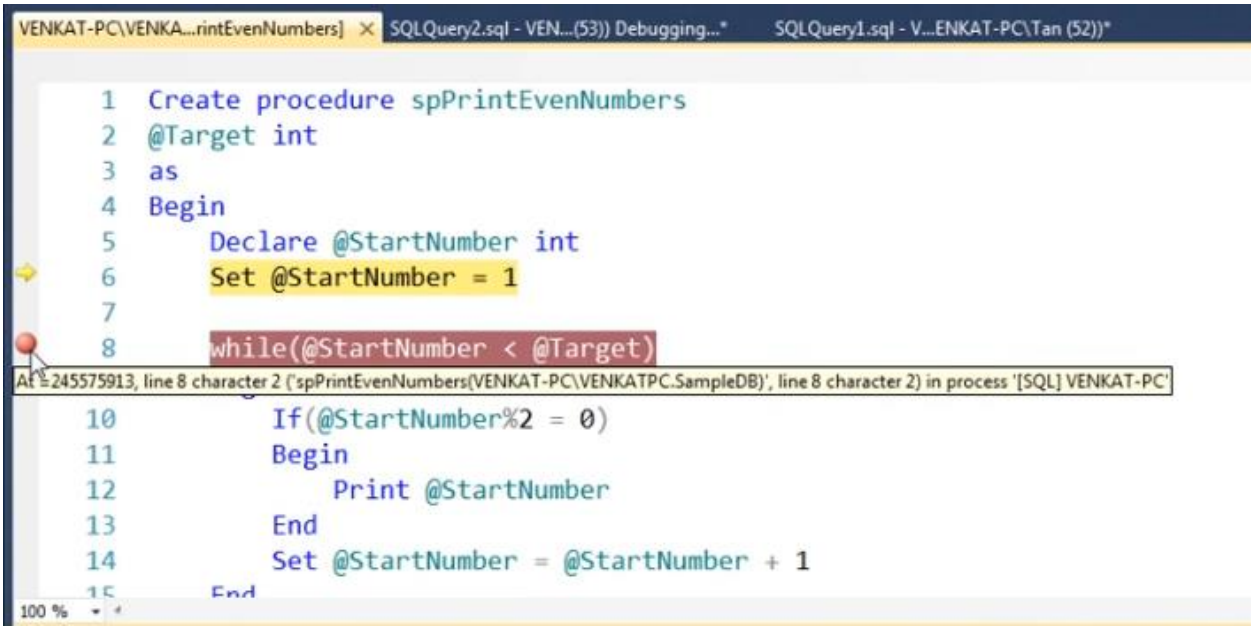


When we click Step Out the debugger finishes debugging.

Setting break point:

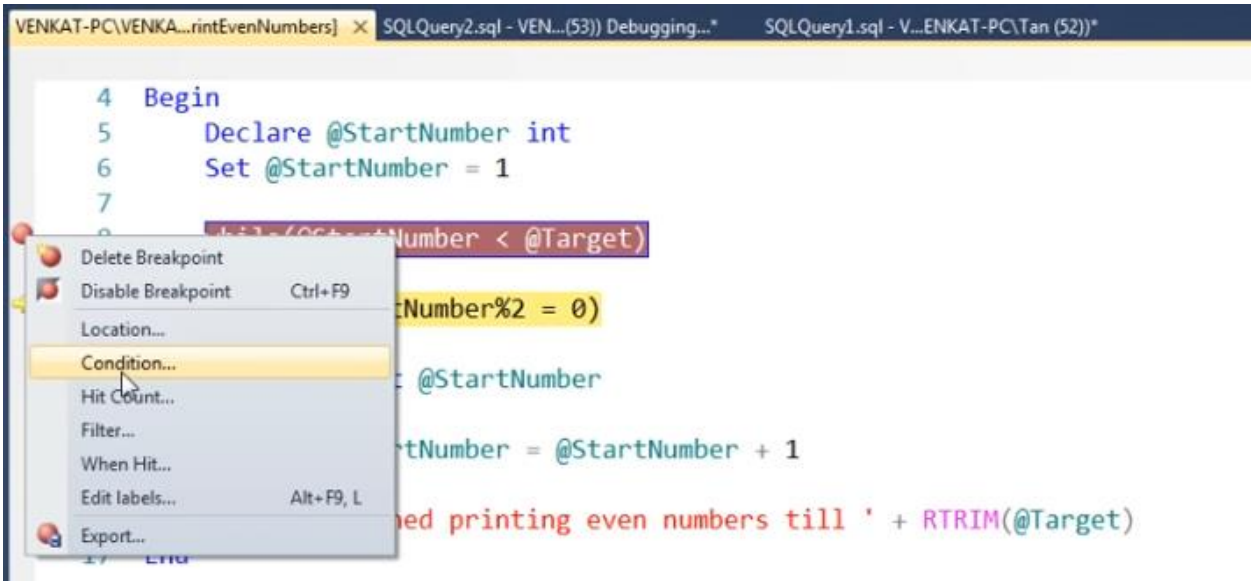
A **breakpoint** is one of the most commonly used **debugging** tools in any programming language. **Breakpoint** lets you specify the location where you want your debugger to stop executing the code.

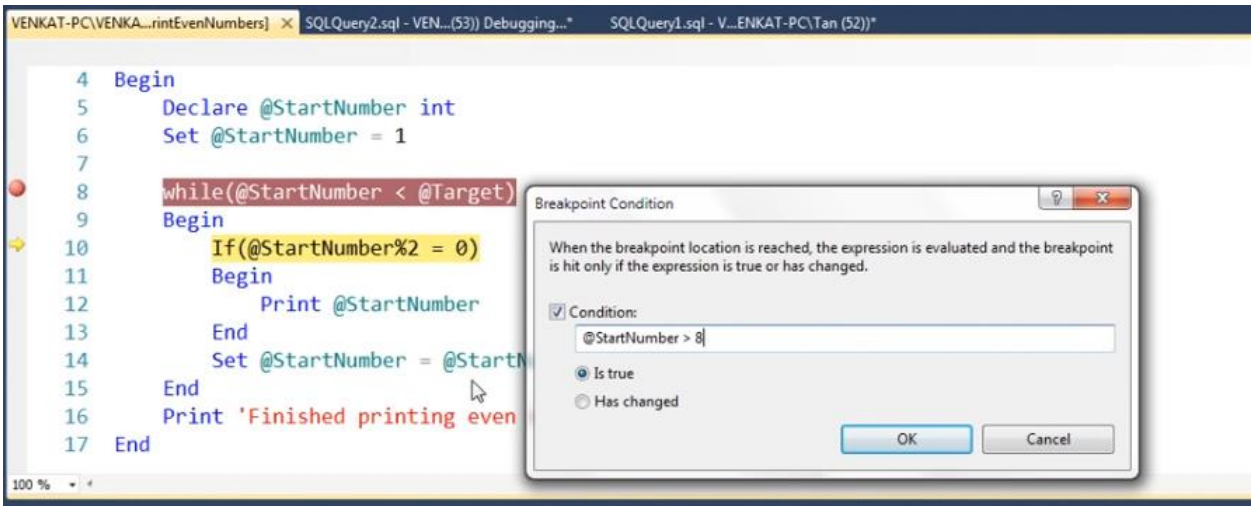




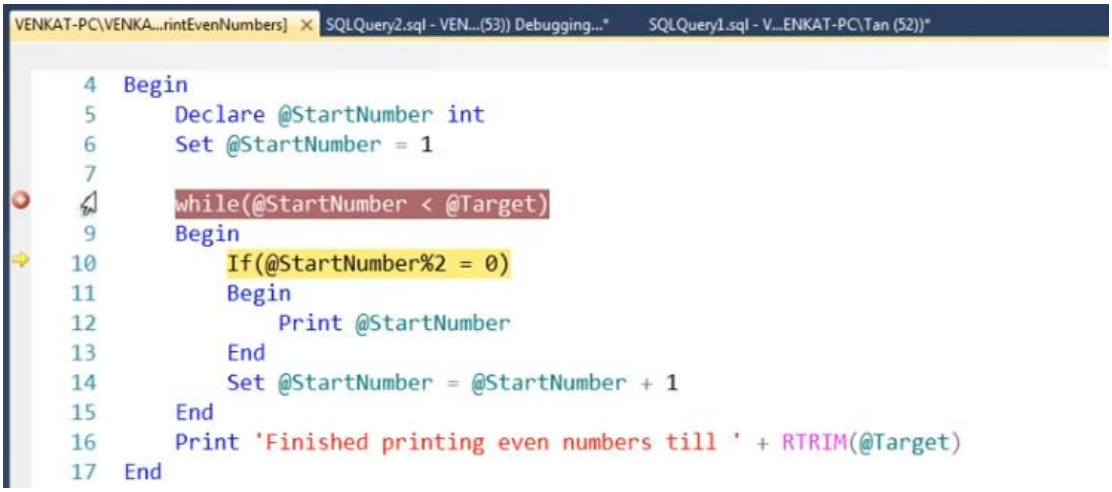
Conditional breakpoint:

If we want to temporarily stop debugging under certain condition at some line of code then we can set conditional breakpoint. To do so, we right click the breakpoint and set the condition as following;





A plus sign appears in the red circle that says its a conditional breakpoint.



Over clause

Over clause in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

The OVER clause combined with PARTITION BY is used to break up data into partitions. The specified function operates for each partition

Syntax: `function (...) OVER (PARTITION BY col1, Col2, ...)`

For example: `COUNT(Gender) OVER (PARTITION BY Gender)` will partition the data by GENDER i.e there will 2 partitions (Male and Female) and then the COUNT() function is applied over each partition

Any of the following functions can be used

COUNT(), AVG(), SUM(), MIN(), MAX(), ROW_NUMBER(), RANK(), DENSE_RANK() etc

Over clause in SQL Server

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

```
SELECT Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

Gender	GenderTotal	AvgSal	MinSal	MaxSal
Female	4	5375	4000	7000
Male	6	5000	3500	6500

What we want to do is that based on given table we want to write a query which is going to give us aggregated data by Gender column.

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```sql
SELECT Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

Results | Messages

	Gender	GenderTotal	AvgSal	MinSal	MaxSal
1	Female	4	5375	4000	7000
2	Male	6	5000	3500	6500

But what if we want that with aggregated data we also want non-aggregate values like name and salary.

Over clause in SQL Server

What if we want non-aggregated values (like employee Name and Salary) in result set along with aggregated values

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Salary	Gender	GenderTotals	AvgSal	MinSal	MaxSal
Pam	5500	Female	4	5375	4000	7000
Sara	4000	Female	4	5375	4000	7000
Mary	5000	Female	4	5375	4000	7000
Jodi	7000	Female	4	5375	4000	7000
Tom	5500	Male	6	5000	3500	6500
Ron	5000	Male	6	5000	3500	6500
Ben	6500	Male	6	5000	3500	6500
Todd	3500	Male	6	5000	3500	6500
Mark	5000	Male	6	5000	3500	6500
John	4500	Male	6	5000	3500	6500

```sql
SELECT Name, Salary, Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

Column 'Employees.Name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause

CAN ACHIEVE THE SAME RESULT USING THIS QUERY WITH OVER CLAUSE:

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```sql
Select Name, Salary, Gender,
COUNT(Gender) Over (Partition By Gender) AS GendersTotal,
AVG(Salary) Over (Partition By Gender) AS AvgSal,
MIN(Salary) Over (Partition By Gender) AS MinSal,
MAX(Salary) Over (Partition By Gender) AS MaxSal
From Employees
```

Results | Messages

	Name	Salary	Gender	GendersTotal	AvgSal	MinSal	MaxSal
1	Pam	5500	Female	4	5375	4000	7000
2	Sara	4000	Female	4	5375	4000	7000
3	Mary	5000	Female	4	5375	4000	7000
4	Jodi	7000	Female	4	5375	4000	7000
5	Tom	5500	Male	6	5000	3500	6500
6	Ron	5000	Male	6	5000	3500	6500
7	Ben	6500	Male	6	5000	3500	6500
8	Todd	3500	Male	6	5000	3500	6500
9	Mark	5000	Male	6	5000	3500	6500
10	John	4500	Male	6	5000	3500	6500

Over clause in SQL Server

One way to achieve this is by including the aggregations in a sub query and then JOINING it with the main query

```
SELECT Name, Salary, Employees.Gender, Genders.GenderTotals,
       Genders.AvgSal, Genders.MinSal, Genders.MaxSal
FROM Employees
INNER JOIN
(SELECT Gender, COUNT(*) AS GenderTotals,
        AVG(Salary) AS AvgSal,
        MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender) AS Genders
ON Genders.Gender = Employees.Gender
```

Better way of doing this is by using the OVER clause combined with PARTITION BY

```
SELECT Name, Salary, Gender,
       COUNT(Gender) OVER(PARTITION BY Gender) AS GenderTotals,
       AVG(Salary) OVER(PARTITION BY Gender) AS AvgSal,
       MIN(Salary) OVER(PARTITION BY Gender) AS MinSal,
       MAX(Salary) OVER(PARTITION BY Gender) AS MaxSal
FROM Employees
```

Row Number Function

ROW_NUMBER function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Row_Number function

- Introduced in SQL Server 2005
- Returns the sequential number of a row starting at 1
- ORDER BY clause is required
- PARTITION BY clause is optional
- When the data is partitioned, row number is reset to 1 when the partition changes

Syntax : `ROW_NUMBER() OVER (ORDER BY Col1, Col2)`

SQLQuery3.sql - V...ENKAT-PC\Tan (52))*

```
Select Name, Gender, Salary,
ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber
from Employees
```

Results | Messages

	Name	Gender	Salary	RowNumber
1	Pam	Female	5500	1
2	Sara	Female	4000	2
3	Mary	Female	5000	3
4	Jodi	Female	7000	4
5	Tom	Male	5500	5
6	Ron	Male	5000	6
7	Ben	Male	6500	7
8	Todd	Male	3500	8
9	Mark	Male	5000	9
10	John	Male	4500	10

If we don't provide Order By clause then it gives error:

```
Select Name, Gender, Salary,
ROW_NUMBER() OVER () AS RowNumber
from Employees
```

Messages

```
Msg 4112, Level 15, State 1, Line 2
The function 'ROW_NUMBER' must have an OVER clause with ORDER BY.
```

SQLQuery3.sql - V...ENKAT-PC\Tan (52))*

```
Select Name, Gender, Salary,
ROW_NUMBER() OVER (PARTITION BY GENDER ORDER BY Gender) AS RowNumber
from Employees
```

Results | Messages

	Name	Gender	Salary	RowNumber
1	Pam	Female	5500	1
2	Sara	Female	4000	2
3	Mary	Female	5000	3
4	Jodi	Female	7000	4
5	Tom	Male	5500	1
6	Ron	Male	5000	2
7	Ben	Male	6500	3
8	Todd	Male	3500	4
9	Mark	Male	5000	5
10	John	Male	4500	6

ROW_NUMBER function in SQL Server

Row_Number function without PARTITION BY : In this example, data is not partitioned, so ROW_NUMBER will provide a consecutive numbering for all the rows in the table based on the order of rows imposed by the ORDER BY clause.

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	5
Ron	Male	5000	6
Ben	Male	6500	7
Todd	Male	3500	8
Mark	Male	5000	9
John	Male	4500	10

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber
FROM   Employees
```

Please note : If ORDER BY clause is not specified you will get the following error

```
The function 'ROW_NUMBER' must have an OVER clause with ORDER BY
```

ROW_NUMBER function in SQL Server

Row_Number function with PARTITION BY : In this example, data is partitioned by Gender, so ROW_NUMBER will provide a consecutive numbering only for the rows with in a parttion. When the partition changes the row number is reset to 1

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	1
Ron	Male	5000	2
Ben	Male	6500	3
Todd	Male	3500	4
Mark	Male	5000	5
John	Male	4500	6

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (PARTITION BY Gender ORDER BY Gender) AS RowNumber
FROM   Employees
```

Use case for Row_Number function : Deleting all duplicate rows except one

Use case for Row_Number function : Deleting all duplicate rows except one

Results | Messages

	ID	FirstName	LastName	Gender	Salary	RowNumber
1	1	Mark	Hastings	Male	60000	1
2	1	Mark	Hastings	Male	60000	2
3	1	Mark	Hastings	Male	60000	3
4	2	Mary	Lambeth	Female	30000	1
5	2	Mary	Lambeth	Female	30000	2
6	3	Ben	Hoskins	Male	70000	1
7	3	Ben	Hoskins	Male	70000	2
8	3	Ben	Hoskins	Male	70000	3

SQLQuery3.sql - V...ENKAT-PC\Tan (52))*

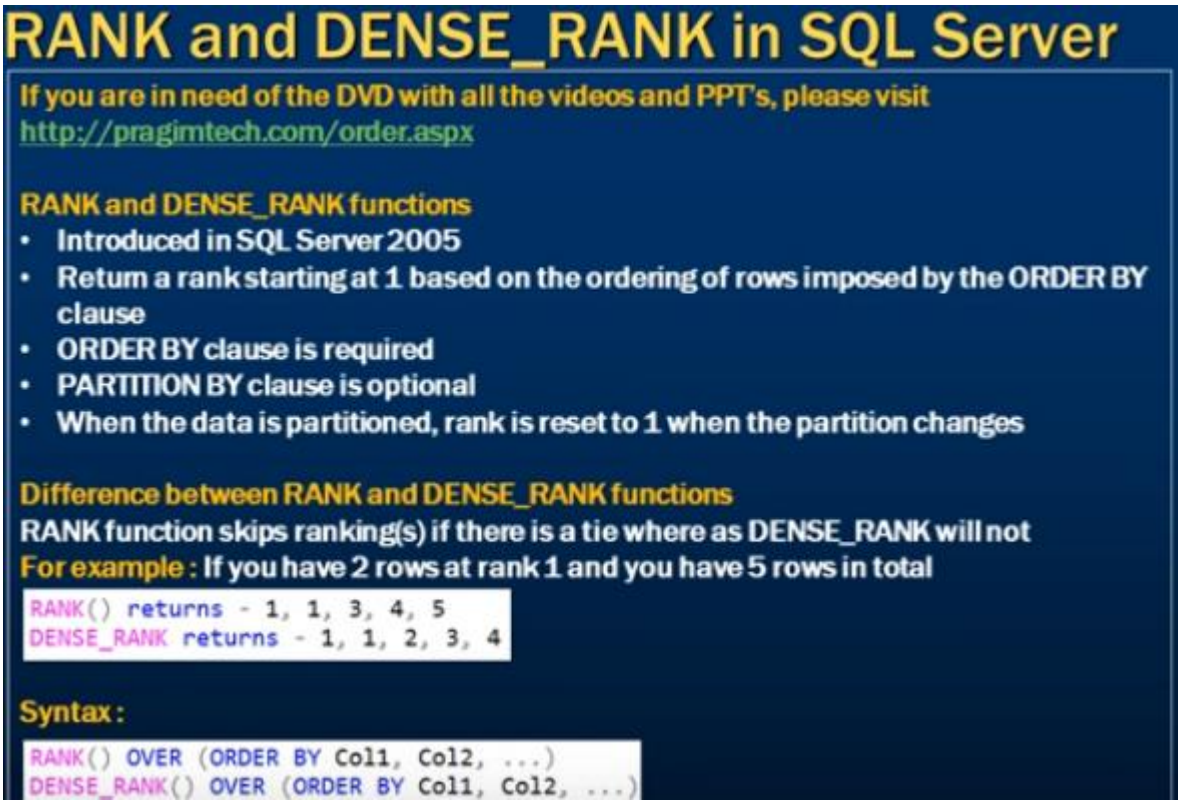
```
WITH EmployeesCTE AS
(
    SELECT *, ROW_NUMBER()OVER(PARTITION BY ID ORDER BY ID) AS RowNumber
    FROM Employees
)
DELETE FROM EmployeesCTE WHERE RowNumber > 1


Select * from Employees
```

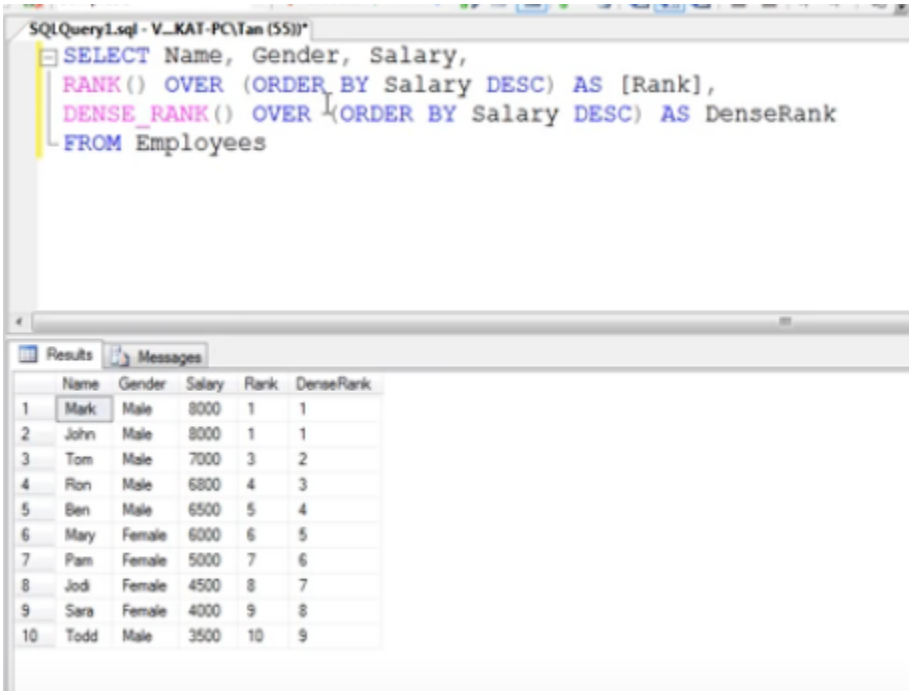
Results | Messages

	ID	FirstName	LastName	Gender	Salary
1	1	Mark	Hastings	Male	60000
2	2	Mary	Lambeth	Female	30000
3	3	Ben	Hoskins	Male	70000

Rank and Dense Rank

The **RANK**() **function** is a window **function** that assigns a **rank** to each row within a partition of a result set. The rows within a partition that have the same values will receive the same **rank**. The **rank** of the first row within a partition is one.





	Name	Gender	Salary	Rank	DenseRank
1	Mark	Male	8000	1	1
2	John	Male	8000	1	1
3	Tom	Male	7000	3	2
4	Ron	Male	6800	4	3
5	Ben	Male	6500	5	4
6	Mary	Female	6000	6	5
7	Pam	Female	5000	7	6
8	Jodi	Female	4500	8	7
9	Sara	Female	4000	9	8
10	Todd	Male	3500	10	9

RANK() and DENSE_RANK() functions without PARTITION BY clause : In this example, data is not partitioned, so RANK() function provides a consecutive numbering except when there is a tie. Rank 2 is skipped as there are 2 rows at rank 1. The third row gets rank 3. DENSE_RANK() on the other hand will not skip ranks if there is a tie. The first 2 rows get rank 1. Third row gets rank 2.

```
SELECT Name, Salary, Gender,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	5000
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	6000
7	Ben	Male	6500
8	Jodi	Female	4500
9	Tom	Male	7000
10	Ron	Male	6800

Name	Salary	Gender	Rank	DenseRank
Mark	8000	Male	1	1
John	8000	Male	1	1
Tom	7000	Male	3	2
Ron	6800	Male	4	3
Ben	6500	Male	5	4
Mary	6000	Female	6	5
Pam	5000	Female	7	6
Jodi	4500	Female	8	7
Sara	4000	Female	9	8
Todd	3500	Male	10	9

RANK() and DENSE_RANK() functions with PARTITION BY clause : Notice when the partition changes from Female to Male Rank is reset to 1

```
SELECT Name, Salary, Gender,
RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	5000
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	6000
7	Ben	Male	6500
8	Jodi	Female	4500
9	Tom	Male	7000
10	Ron	Male	6800

Name	Salary	Gender	Rank	DenseRank
Mary	6000	Female	1	1
Pam	5000	Female	2	2
Jodi	4500	Female	3	3
Sara	4000	Female	4	4
Mark	8000	Male	1	1
John	8000	Male	1	1
Tom	7000	Male	3	2
Ron	6800	Male	4	3
Ben	6500	Male	5	4
Todd	3500	Male	6	5

RANK() FUNCTION SKIPS THE NEXT CONSECUTIVE RANKING NUMBER IF THERE IS A TIE WHEREAS DENSERANK() FUNCTION DOES NOT.

TO FIND HIGHEST SALARY OF AN EMPLOYEE

Use case for RANK and DENSE_RANK functions : Both these functions can be used to find Nth highest salary. However, which function to use depends on what you want to do when there is a tie.

Since we have 2 Employees with the FIRST highest salary. Rank() function will not return any rows for the SECOND highest Salary.

```
WITH Result AS
(
    SELECT Salary, RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

Though we have 2 Employees with the FIRST highest salary. Dense_Rank() function returns, the next Salary after the tied rows as the SECOND highest Salary

```
WITH Result AS
(
    SELECT Salary, DENSE_RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

You can also use RANK and DENSE_RANK functions to find the Nth highest Salary among Male or Female employee groups.

The following query finds the 3rd highest salary amount paid among the Female employees group

```
WITH Result AS
(
    SELECT Salary, Gender,
           DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 3 AND Gender = 'Female'
```

Difference between rank dense rank and row number

RANK, DENSE_RANK & ROW_NUMBER

You will only see the difference when there are ties

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	8000
4	Sara	Female	4000
5	Todd	Male	3500

Name	Salary	Gender	RowNumber	Rank	DenseRank
Mark	8000	Male	1	1	1
John	8000	Male	2	1	1
Pam	8000	Female	3	1	1
Sara	4000	Female	4	4	2
Todd	3500	Male	5	5	3

Difference between RANK, DENSE_RANK and ROW_NUMBER functions

- ROW_NUMBER : Returns an increasing unique number for each row starting at 1, even if there are duplicates.
- RANK : Returns an increasing unique number for each row starting at 1. When there are duplicates, same rank is assigned to all the duplicate rows, but the next row after the duplicate rows will have the rank it would have been assigned if there had been no duplicates. So RANK function skips rankings if there are duplicates.
- DENSE_RANK : Returns an increasing unique number for each row starting at 1. When there are duplicates, same rank is assigned to all the duplicate rows but the DENSE_RANK function will not skip any ranks. This means the next row after the duplicate rows will have the next rank in the sequence.

SQLQuery1.sql - V...ENKAT-PC\Tan (52))* | SQLQuery2.sql - V...ENKAT-PC\Tan (53))*

```sql
SELECT Name, Salary, Gender,
ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

100 %

Results | Messages

	Name	Salary	Gender	RowNumber	Rank	DenseRank
1	Mark	8000	Male	1	1	1
2	John	8000	Male	2	1	1
3	Pam	8000	Female	3	1	1
4	Sara	4000	Female	4	4	2
5	Todd	3500	Male	5	5	3

Calculate running total

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

SQL Query to compute running total without partitions

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RunningTotal
Mark	Male	5000	5000
John	Male	4500	9500
Pam	Female	5500	15000
Sara	Female	4000	19000
Todd	Male	3500	22500
Mary	Female	5000	27500
Ben	Male	6500	34000
Jodi	Female	7000	41000
Tom	Male	5500	46500
Ron	Male	5000	51500

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY ID) AS RunningTotal
FROM Employees
```

CAN ALSO PARTITION BY GENDER

SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
Select Name, Gender, Salary,
Sum(Salary) Over (Partition By Gender Order By Id) as RunningTotal
From Employees
```

100 %

Results | Messages

	Name	Gender	Salary	RunningTotal
1	Pam	Female	5500	5500
2	Sara	Female	4000	9500
3	Mary	Female	5000	14500
4	Jodi	Female	7000	21500
5	Mark	Male	5000	5000
6	John	Male	4500	9500
7	Todd	Male	3500	13000
8	Ben	Male	6500	19500
9	Tom	Male	5500	25000
10	Ron	Male	5000	30000

IF WE ORDER BY SALARY WHICH HAS DUPLICATE ROWS

SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
Select * from Employees Order By Salary

Select Name, Gender, Salary,
Sum(Salary) Over (Order By Salary) as RunningTotal
From Employees
```

100 %

Results | Messages

	Name	Gender	Salary	RunningTotal
1	Todd	Male	3500	3500
2	Sara	Female	4000	7500
3	John	Male	4500	12000
4	Mark	Male	5000	27000
5	Mary	Female	5000	27000
6	Ron	Male	5000	27000
7	Tom	Male	5500	38000
8	Pam	Female	5500	38000
9	Ben	Male	6500	44500
10	Jodi	Female	7000	51500

Running Total in SQL Server 2012

What happens if I use order by on Salary column

If you have duplicate values in the Salary column, all the duplicate values will be adde to the running total at once. In the example below notice that we have 5000 repeated times. So 15000 (i.e 5000 + 5000 + 5000) is added to the running total at once

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RunningTotal
Todd	Male	3500	3500
Sara	Female	4000	7500
John	Male	4500	12000
Mark	Male	5000	27000
Mary	Female	5000	27000
Ron	Male	5000	27000
Tom	Male	5500	38000
Pam	Female	5500	38000
Ben	Male	6500	44500
Jodi	Female	7000	51500

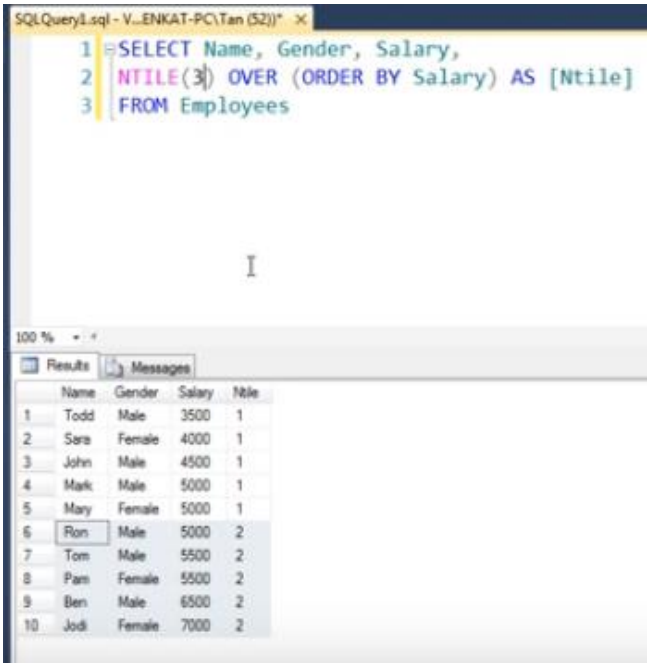
```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY Salary) AS RunningTotal
FROM Employees
```

So when computing running total, it is better to use a column that has unique data in the ORDER BY clause.

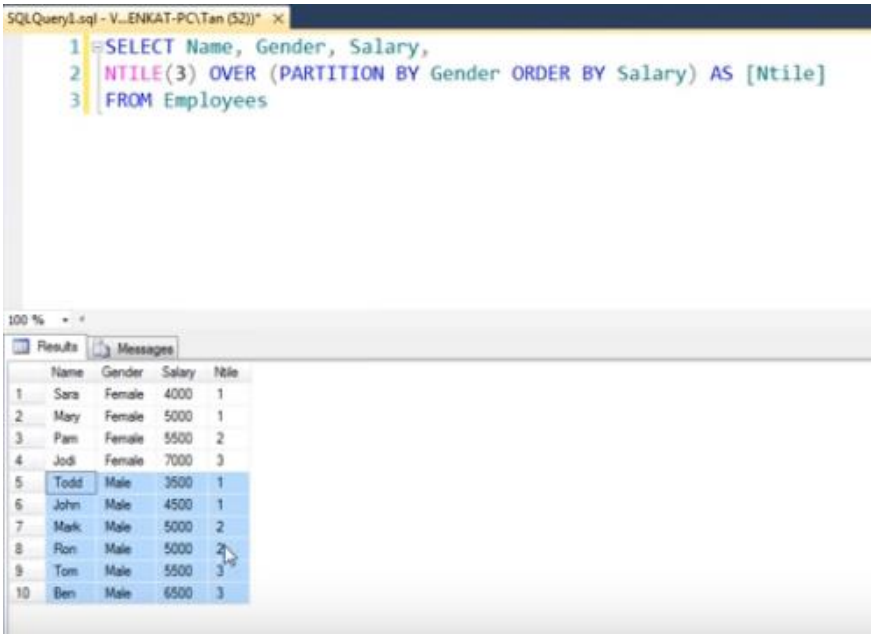
NTILE function

The **SQL NTILE**() is a window **function** that allows you to break the result set into a specified number of approximately equal groups, or buckets. It assigns each group a bucket number starting from one. For each row in a group, the **NTILE**() **function** assigns a bucket number representing the group to which the row belongs.

NTILE() FUNCTION DIVIDES THE ROWS IN TO 3 GROUPS:



IF WE PARTITION BY GENDER, WE CAN STILL FORM 3 GROUPS IN EACH GENDER USING NTILE():



Lead and Lag functions

Lead & Lag functions in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Lead and Lag functions

- Introduced in SQL Server 2012
- Lead function is used to access subsequent row data along with current row data
- Lag function is used to access previous row data along with current row data
- ORDER BY clause is required
- PARTITION BY clause is optional

Syntax:

```
LEAD(Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
LAG (Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
```

Offset	Number of rows to lead or lag
Default_Value	The default value to return if the number of rows to lead or lag goes beyond first row or last row in a table or partition. If default value is not specified NULL is returned.

LEAD FUNCTION ALLOWS US TO LOOK DOWN A COLUMN AND RETRIEVE THE NEXT VALUE IN THE ROW TO FILL IN TO THE LEAD COLUMN:

THE DEFAULT PARAMETER FOR LEAD FUNCTION IS 1 WHICH MEANS TO LOOK DOWN 1 ROW

SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

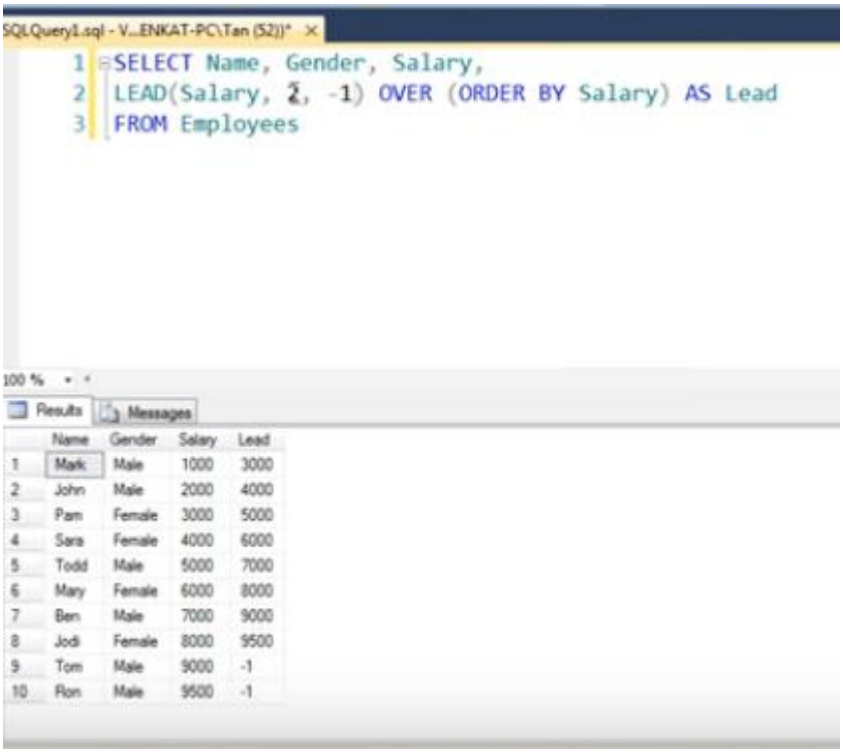
```
SELECT Name, Gender, Salary,
LEAD(Salary) OVER (ORDER BY Salary) AS Lead
FROM Employees
```

100 %

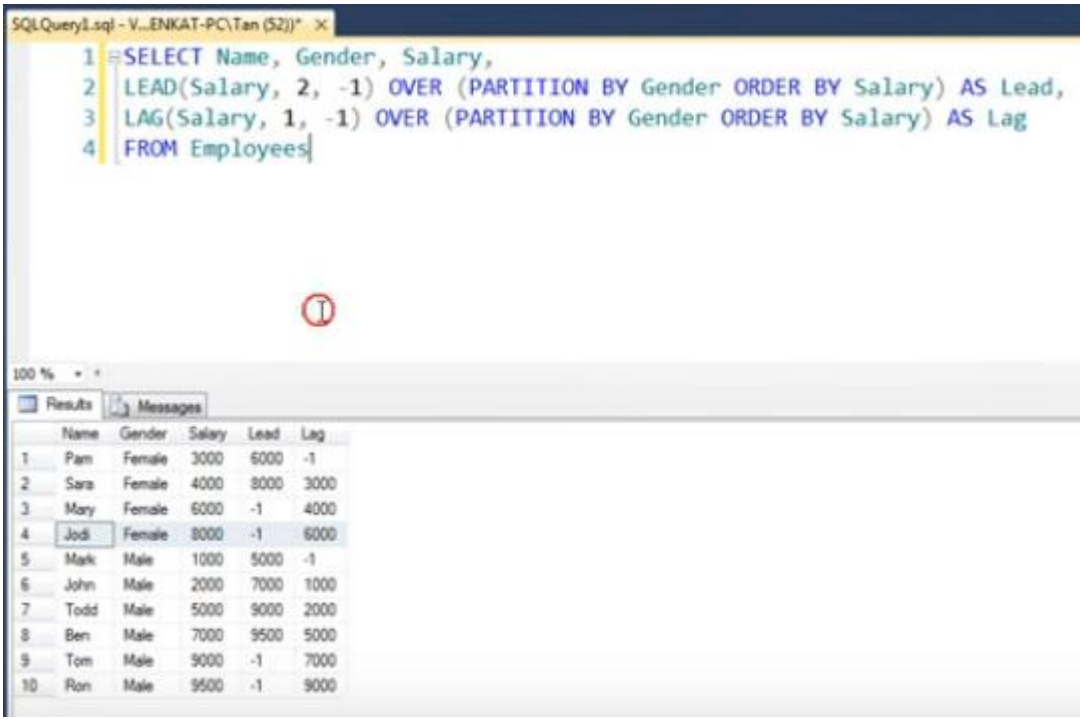
Results | Messages

	Name	Gender	Salary	Lead
1	Mark	Male	1000	2000
2	John	Male	2000	3000
3	Pam	Female	3000	4000
4	Sara	Female	4000	5000
5	Todd	Male	5000	6000
6	Mary	Female	6000	7000
7	Ben	Male	7000	8000
8	Jodi	Female	8000	9000
9	Tom	Male	9000	9500
10	Ron	Male	9500	NULL

WHEN WE USE 2 AS THE PARAMETER IN LEAD FUNCTION, THEN WE LOOK DOWN 2 ROWS AND -1 SIGNIFIES THAT WE WANT TO LOOK DOWN:



LAG FUNCTION ALSO WORKS IN SIMILAR FASHION BUT INSTEAD OF LOOKING FORWARD IT WILL LOOK BAD: CAN ALSO PARTITION BY GENDER



FIRST VALUE function

FIRST_VALUE function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

FIRST_VALUE function

- Introduced in SQL Server 2012
- Retrieves the first value from the specified column
- ORDER BY clause is required
- PARTITION BY clause is optional

Syntax:

```
FIRST_VALUE(Column_Name) OVER (ORDER BY Col1, Col2, ...)
```

SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary) AS FirstValue
FROM Employees
```

Results | Messages

	Name	Gender	Salary	FirstValue
1	Pam	Female	3000	Pam
2	Sara	Female	4000	Pam
3	Mary	Female	6000	Pam
4	Jodi	Female	8000	Pam
5	Mark	Male	1000	Mark
6	John	Male	2000	Mark
7	Todd	Male	5000	Mark
8	Ben	Male	7000	Mark
9	Tom	Male	9000	Mark
10	Ron	Male	9500	Mark

FIRST_VALUE function example WITH partitions : Returns the name of the lowest paid employee from the respective partition

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary) AS FirstValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

→

Name	Gender	Salary	FirstValue
Pam	Female	3000	Pam
Sara	Female	4000	Pam
Mary	Female	6000	Pam
Jodi	Female	8000	Pam
Mark	Male	1000	Mark
John	Male	2000	Mark
Todd	Male	5000	Mark
Ben	Male	7000	Mark
Tom	Male	9000	Mark
Ron	Male	9500	Mark

Window functions

Window functions in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Different categories of window functions

- Aggregate functions - AVG, SUM, COUNT, MIN, MAX etc..
- Ranking functions - RANK, DENSE_RANK, ROW_NUMBER etc..
- Analytic functions - LEAD, LAG, FIRST_VALUE, LAST_VALUE etc...

OVER Clause defines the partitioning and ordering of rows (i.e a window) for the above functions to operate on. Hence these functions are called window functions. The OVER clause accepts the following three arguments to define a window for these functions to operate on

- ORDER BY : Defines the logical order of the rows
- PARTITION BY : Divides the query result set into partitions. The window function is applied to each partition separately.
- ROWS or RANGE clause : Further limits the rows within the partition by specifying start and end points within the partition.

The default for ROWS or RANGE clause is

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

SQLQuery2.sql - V...ENKAT-PC\Tan (52))*

```sql
SELECT Name, Gender, Salary,
AVG(Salary) OVER (ORDER BY Salary) AS Average,
COUNT(Salary) OVER (ORDER BY Salary) AS [Count],
SUM(Salary) OVER (ORDER BY Salary) AS [Sum]
FROM Employees


Select AVG(Salary)
from Employees
```

100 %

Results | Messages

	Name	Gender	Salary	Average	Count	Sum
1	Mark	Male	1000	1000	1	1000
2	John	Male	2000	1500	2	3000
3	Pam	Female	3000	2000	3	6000
4	Sara	Female	4000	2500	4	10000
5	Todd	Male	5000	3000	5	15000
6	Mary	Female	6000	3500	6	21000
7	Ben	Male	7000	4000	7	28000
8	Jodi	Female	8000	4500	8	36000
9	Tom	Male	9000	5000	9	45000
10	Ron	Male	9500	5450	10	54500

THE FOLLOWING CODE FURTHER LIMITS THE ROWS WITHIN THE PARTITION BY SPECIFYING THE START AND END POINT WITHIN THE PARTITION:

The default for ROWS or RANGE clause is

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

SQLQuery2.sql - V...ENKAT-PC\Tan (52))*

```
SELECT Name, Gender, Salary,
AVG(Salary) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Averag
COUNT(Salary) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS [Cou
SUM(Salary) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS [Sum]
FROM Employees


Select AVG(Salary)
from Employees
```

100 %

Results | Messages

	Name	Gender	Salary	Average	Count	Sum
1	Mark	Male	1000	5450	10	54500
2	John	Male	2000	5450	10	54500
3	Pam	Female	3000	5450	10	54500
4	Sara	Female	4000	5450	10	54500
5	Todd	Male	5000	5450	10	54500
6	Mary	Female	6000	5450	10	54500
7	Ben	Male	7000	5450	10	54500
8	Jodi	Female	8000	5450	10	54500
9	Tom	Male	9000	5450	10	54500
10	Ron	Male	9500	5450	10	54500

TAKES AVG, SUM, COUNT BETWEEN 1 PRECEDING AND 1 FOLLOWING ROW

SQLQuery2.sql - V...ENKAT-PC\Tan (52))*

```
SELECT Name, Gender, Salary,
AVG(Salary) OVER (ORDER BY Salary ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS Average,
COUNT(Salary) OVER (ORDER BY Salary ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS [Count],
SUM(Salary) OVER (ORDER BY Salary ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS [Sum]
FROM Employees


Select AVG(Salary)
from Employees
```

100 %

Results | Messages

	Name	Gender	Salary	Average	Count	Sum
1	Mark	Male	1000	1500	2	3000
2	John	Male	2000	2000	3	6000
3	Pam	Female	3000	3000	3	9000
4	Sara	Female	4000	4000	3	12000
5	Todd	Male	5000	5000	3	15000
6	Mary	Female	6000	6000	3	18000
7	Ben	Male	7000	7000	3	21000
8	Jodi	Female	8000	8000	3	24000
9	Tom	Male	9000	8833	3	26500
10	Ron	Male	9500	9250	2	18500

Difference between rows and range

Difference between ROWS and RANGE

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Calculate the running total of Salary and display it against every employee row

Id	Name	Salary
1	Mark	1000
2	John	2000
3	Pam	3000
4	Sara	4000
5	Todd	3000

Name	Salary	RunningTotal
Mark	1000	1000
John	2000	3000
Pam	3000	6000
Sara	4000	10000
Todd	5000	15000

ROWS or RANGE clause is not specified. It's default value is used.

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary) AS RunningTotal
FROM Employees
```

The above query can be re-written using an explicit value for ROWS or RANGE clause

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

```
SELECT Name, Salary,
SUM(Salary) OVER (ORDER BY Salary) AS RunningTotal
FROM Employees
```

	Name	Salary	RunningTotal
1	Mark	1000	1000
2	John	2000	3000
3	Pam	3000	6000
4	Sara	4000	10000
5	Todd	5000	15000

CAN ALSO GET RUNNING TOTAL USING RANGE FUNCTION:

```
SELECT Name, Salary,
SUM(Salary) OVER (ORDER BY Salary RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

	Name	Salary	RunningTotal
1	Mark	1000	1000
2	John	2000	3000
3	Pam	3000	6000
4	Sara	4000	10000
5	Todd	5000	15000

Difference between ROWS and RANGE

Calculate the running total of Salary and display it against every employee row

Id	Name	Salary
1	Mark	1000
2	John	2000
3	Pam	3000
4	Sara	4000
5	Todd	3000

Name	Salary	RunningTotal
Mark	1000	1000
John	2000	3000
Pam	3000	6000
Sara	4000	10000
Todd	5000	15000

We can also achieve the same result, by replacing RANGE with ROWS

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

What is the difference between ROWS and RANGE

The difference is in the way duplicate rows are treated. ROWS treat duplicates as distinct values, where as RANGE treats them as a single entity.

RANGE() IS TREATING DUPLICATES AS THE SAME

```
Select * from Employees

SELECT Name, Salary,
SUM(Salary) OVER (ORDER BY Salary RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Results | Messages

	Name	Salary	RunningTotal
1	Mark	1000	2000
2	John	1000	2000
3	Pam	3000	8000
4	Sara	3000	8000
5	Todd	5000	13000

BY DEFAULT ORDER BY ALSO WORKS AS -> RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Difference between ROWS and RANGE

Calculate the running total of Salary and display it against every employee row

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Id	Name	Salary
1	Mark	1000
2	John	1000
3	Pam	3000
4	Sara	3000
5	Todd	3000

→

Name	Salary	RunningTotal
Mark	1000	1000
John	1000	2000
Pam	3000	5000
Sara	3000	8000
Todd	5000	13000

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Id	Name	Salary
1	Mark	1000
2	John	1000
3	Pam	3000
4	Sara	3000
5	Todd	3000

→

Name	Salary	RunningTotal
Mark	1000	2000
John	1000	2000
Pam	3000	8000
Sara	3000	8000
Todd	5000	13000

LAST VALUE function

LAST_VALUE. LAST_VALUE is a window function that returns the values of a specified expression that is evaluated at the last row of a window for the current row. This means that you can select a last **value** from a set of rows without having to use a self join.

LAST_VALUE function not working as expected : In the following example, LAST_VALUE function does not return the name of the highest paid employee. This is because we have not specified an explicit value for ROWS or RANGE clause. As a result it is using it's default value `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`

```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (ORDER BY Salary) AS LastValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Mark	Male	1000	Mark
John	Male	2000	John
Pam	Female	3000	Pam
Sara	Female	4000	Sara
Todd	Male	5000	Todd
Mary	Female	6000	Mary
Ben	Male	7000	Ben
Jodi	Female	8000	Jodi
Tom	Male	9000	Tom
Ron	Male	9500	Ron

LAST_VALUE function working as expected : LAST_VALUE function in this case returns the name of the highest paid employee as expected, as we have set an explicit value for ROWS or RANGE clause to `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`

This tells the LAST_VALUE function that it's window starts at the first row and ends at the last row in the result set

```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING
                              AND UNBOUNDED FOLLOWING) AS LastValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Mark	Male	1000	Ron
John	Male	2000	Ron
Pam	Female	3000	Ron
Sara	Female	4000	Ron
Todd	Male	5000	Ron
Mary	Female	6000	Ron
Ben	Male	7000	Ron
Jodi	Female	8000	Ron
Tom	Male	9000	Ron
Ron	Male	9500	Ron

FROM PARTITION WE CAN SEE JODI IS FEMALE EMPLOYEE WITH HIGHEST SALARY AND RON IS MALE EMPLOYEE WITH HIGHEST SALARY

LAST_VALUE function example with partitions : In the following example, LAST_VALUE function returns the name of the highest paid employee from the respective partition

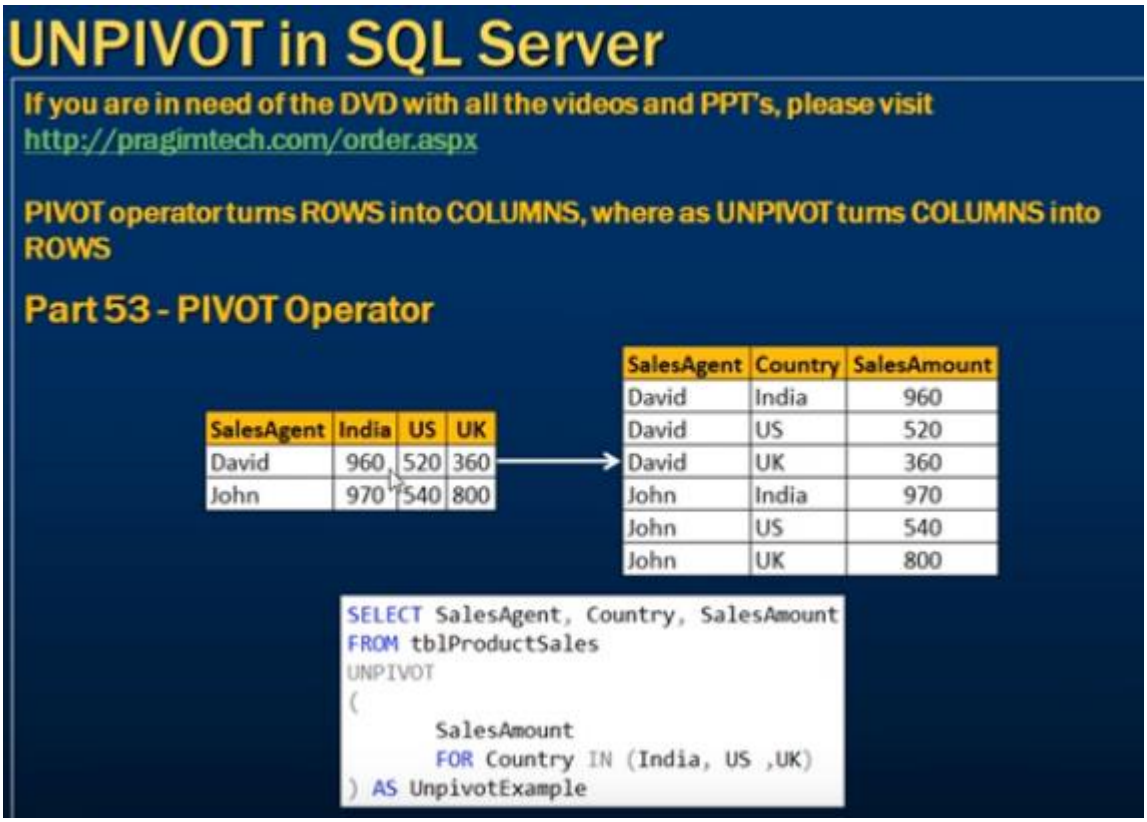
```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary ROWS BETWEEN
                              UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS LastValue
FROM Employees
```

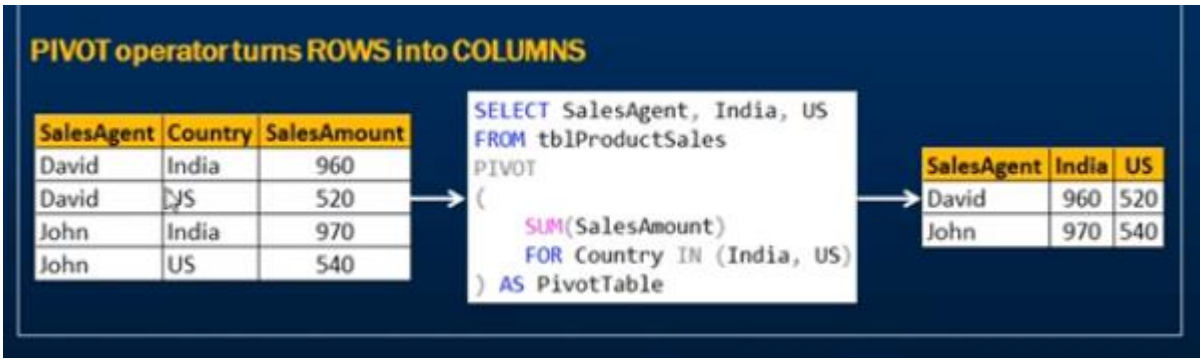
Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Pam	Female	3000	Jodi
Sara	Female	4000	Jodi
Mary	Female	6000	Jodi
Jodi	Female	8000	Jodi
Mark	Male	1000	Ron
John	Male	2000	Ron
Todd	Male	5000	Ron
Ben	Male	7000	Ron
Tom	Male	9000	Ron
Ron	Male	9500	Ron

UNPIVOT

The PIVOT statement is used to convert table rows into columns, while the *UNPIVOT* operator converts columns back to rows. Reversing a PIVOT statement refers to the process of applying the *UNPIVOT* operator to the already PIVOTED dataset in order to retrieve the original dataset.





UNPIVOT operator reverses what PIVOT operator has done (turn COLUMNS into ROWS)

SalesAgent	India	US
David	960	520
John	970	540

```
SELECT SalesAgent, Country, SalesAmount
FROM
(SELECT SalesAgent, India, US
FROM tblProductSales
PIVOT
(
    SUM(SalesAmount)
    FOR Country IN (India, US)
) AS PivotTable) P
UNPIVOT
(
    SalesAmount
    FOR Country IN (India, US)
) AS UnpivotTable
```

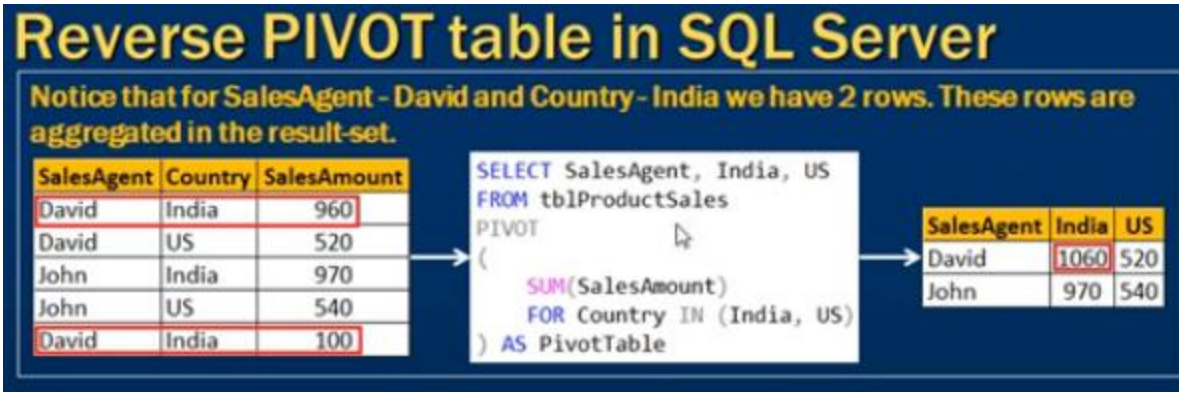
We are able to get the original data back, because the SUM aggregate function that we used with the PIVOT operator did not perform any aggregation

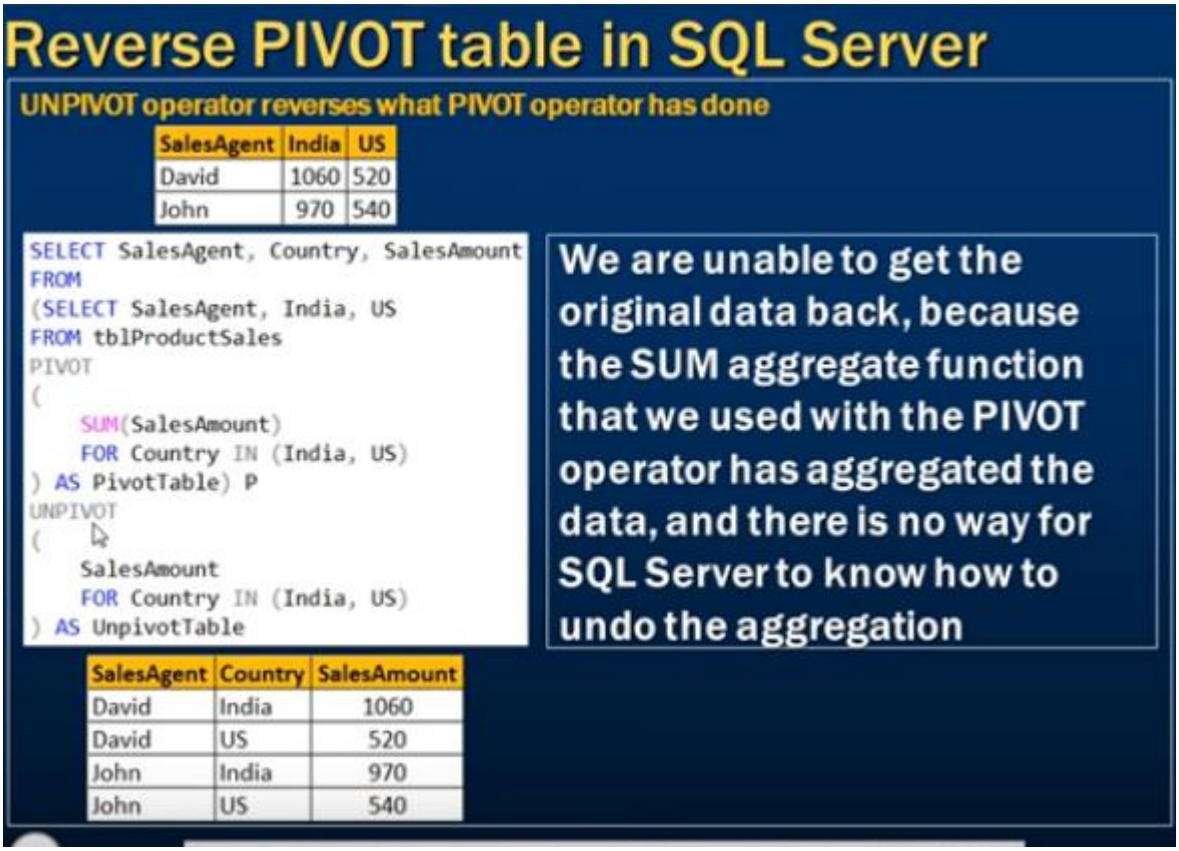
SalesAgent	Country	SalesAmount
David	India	960
David	US	520
John	India	970
John	US	540

Is it always possible to reverse what PIVOT operator has done using UNPIVOT operator

No, not always. If the PIVOT operator has not aggregated the data, you can get your original data back using the UNPIVOT operator but not if the data is aggregated.

Reverse PIVOT





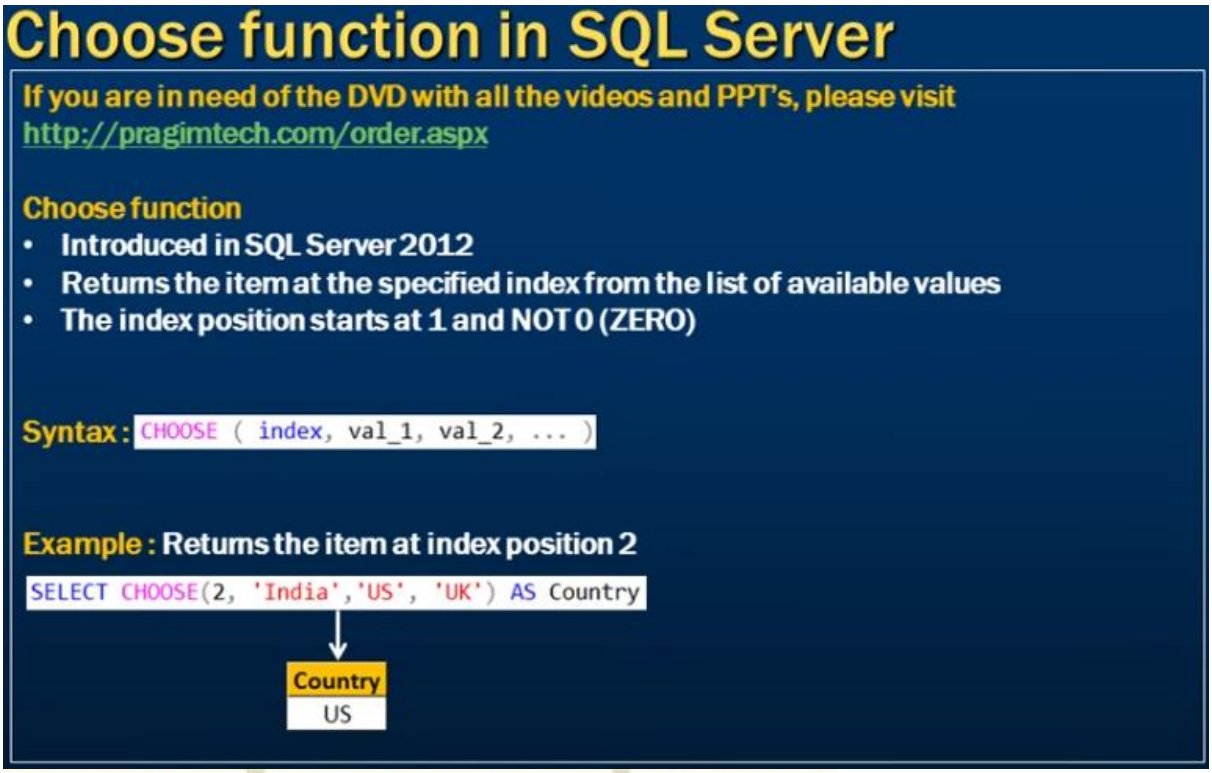
```sql
SELECT Name, DateOfBirth,
CASE DATEPART(MM, DateOfBirth)
    WHEN 1 THEN 'JAN'
    WHEN 2 THEN 'FEB'
    WHEN 3 THEN 'MAR'
    WHEN 4 THEN 'APR'
    WHEN 5 THEN 'MAY'
    WHEN 6 THEN 'JUN'
    WHEN 7 THEN 'JUL'
    WHEN 8 THEN 'AUG'
    WHEN 9 THEN 'SEP'
    WHEN 10 THEN 'OCT'
    WHEN 11 THEN 'NOV'
    WHEN 12 THEN 'DEC'
END AS [Month]
FROM Employees
```

Results | Messages

	Name	DateOfBirth	Month
1	Mark	1980-01-11	JAN
2	John	1981-12-12	DEC
3	Amy	1979-11-21	NOV
4	Ben	1978-05-14	MAY
5	Sara	1970-03-17	MAR
6	David	1978-04-05	APR

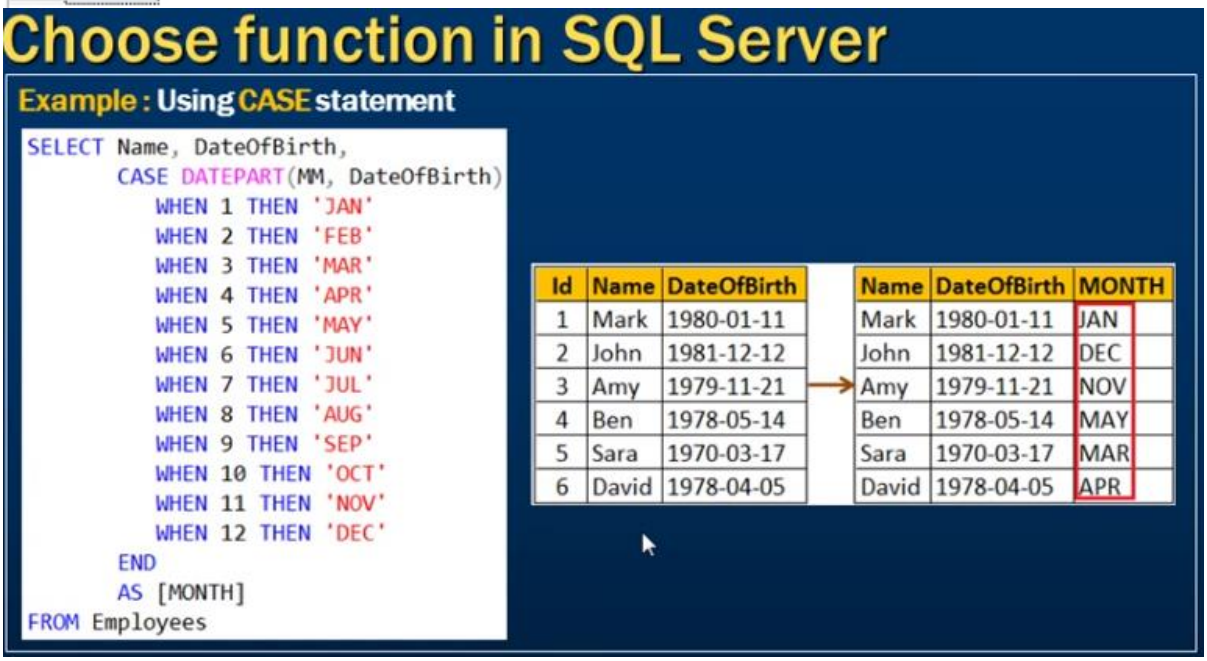
Choose function

The **CHOOSE**() **function** returns the item from a list of items at a specified index. In this syntax: index is an expression integer that specifies the index of the element to be returned. Note that the indexes of the elements are 1-based.




```
SELECT CHOOSE(2, 'India','US','UK') AS Country
```

	Country
1	US



```sql
SELECT Name, DateOfBirth,
CHOOSE(DATEPART(MM, DateOfBirth), 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT
FROM Employees
```

Results | Messages

	Name	DateOfBirth	Month
1	Mark	1980-01-11	JAN
2	John	1981-12-12	DEC
3	Amy	1979-11-21	NOV
4	Ben	1978-05-14	MAY
5	Sara	1970-03-17	MAR
6	David	1978-04-05	APR

Using CHOOSE function

```sql
SELECT Name, DateOfBirth,CHOOSE(DATEPART(MM, DateOfBirth),
       'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG',
       'SEP', 'OCT', 'NOV', 'DEC') AS [MONTH]
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	1980-01-11
2	John	1981-12-12
3	Amy	1979-11-21
4	Ben	1978-05-14
5	Sara	1970-03-17
6	David	1978-04-05

Name	DateOfBirth	MONTH
Mark	1980-01-11	JAN
John	1981-12-12	DEC
Amy	1979-11-21	NOV
Ben	1978-05-14	MAY
Sara	1970-03-17	MAR
David	1978-04-05	APR

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```sql
Declare @GenderId INT
SET @GenderId = 1

SELECT IIF(@GenderId = 1, 'Male', 'Female') AS Gender
```

100 %

Results | Messages

	Gender
1	Male

IIF function

Return "YES" if the condition is TRUE, or "NO" if the condition is FALSE: It is a shorthand form of writing CASE statement logic or IF-ELSE logic.

IIF function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

IIF function

- Introduced in SQL Server 2012
- Returns one of two the values, depending on whether the Boolean expression evaluates to true or false
- IIF is a shorthand way for writing a CASE expression

Syntax :

```
IIF ( boolean_expression, true_value, false_value )
```

Example : Returns Male as the boolean expression evaluates to TRUE

```
DECLARE @Gender INT
SET @Gender = 1
SELECT IIF( @Gender = 1, 'Male', 'Femlae') AS Gender
```

Gender
Male

IIF function in SQL Server

Example : Using IIF() function with table data

Id	Name	GenderId
1	Mark	1
2	John	1
3	Amy	2
4	Ben	1
5	Sara	2
6	David	1

Name	GenderId	Gender
Mark	1	Male
John	1	Male
Amy	2	Female
Ben	1	Male
Sara	2	Female
David	1	Male

Using CASE statement

```
SELECT Name, GenderId,
       CASE WHEN GenderId = 1
            THEN 'Male'
            ELSE 'Female'
       END AS Gender
FROM Employees
```

Using IIF function

```
SELECT Name, GenderId,
       IIF(GenderId = 1, 'Male', 'Female') AS Gender
FROM Employees
```

```sql
Declare @GenderId INT
SET @GenderId = 1

SELECT IIF(@GenderId = 1, 'Male', 'Female') AS Gender
```

Results | Messages

	Gender
1	Male

```sql
SELECT Name, GenderId,
CASE WHEN GenderId = 1
     THEN 'Male'
     ELSE 'Female'
END AS Gender
FROM Employees
```

Results | Messages

	Name	GenderId	Gender
1	Mark	1	Male
2	John	1	Male
3	Amy	2	Female
4	Ben	1	Male
5	Sara	2	Female
6	David	1	Male

```sql
SELECT Name, GenderId,
IIF(GenderId = 1, 'Male', 'Female') AS Gender
FROM Employees
```

Results | Messages

	Name	GenderId	Gender
1	Mark	1	Male
2	John	1	Male
3	Amy	2	Female
4	Ben	1	Male
5	Sara	2	Female
6	David	1	Male

TRY PARSE function

- TRY_PARSE function
- Difference between PARSE and TRY_PARSE functions

TRY_PARSE function in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

TRY_PARSE function

- Introduced in SQL Server 2012
- Converts a string to Date/Time or Numeric type
- Returns NULL if the provided string cannot be converted to the specified data type
- Requires .NET Framework Common Language Runtime (CLR)

Syntax : `TRY_PARSE ( string_value AS data_type )`

Example : As the string can be converted to INT, the result will be 99

```
SELECT TRY_PARSE('99' AS INT) AS Result
```

Result
99

Example : As the string cannot be converted to INT, TRY_PARSE returns NULL

```
SELECT TRY_PARSE('ABC' AS INT) AS Result
```

Result
NULL

```
1  SELECT TRY_PARSE('99' AS INT) AS Result
```

Results | Messages

	Result
1	99

```
1  SELECT TRY_PARSE('ABC' AS INT) AS Result
```

Results | Messages

	Result
1	NULL

TRY_PARSE function in SQL Server 2012

Use CASE statement or IIF function to provide a meaningful error message instead of NULL when the conversion fails

Example : Using CASE statement to provide a meaningful error message when the conversion fails

```
SELECT
CASE WHEN TRY_PARSE('ABC' AS INT) IS NULL
     THEN 'Conversion Failed'
     ELSE 'Conversion Successful'
END AS Result
```

Result
Conversion Failed

Example : Using IIF function to provide a meaningful error message when the conversion fails

```
SELECT IIF(TRY_PARSE('ABC' AS INT) IS NULL,
           'Conversion Failed',
           'Conversion Successful') AS Result
```

Result
Conversion Failed

TRY_PARSE function in SQL Server 2012

What is the difference between PARSE and TRY_PARSE

PARSE will result in an error if the conversion fails, where as TRY_PARSE will return NULL instead of an error

Since ABC cannot be converted to INT, PARSE will return an error

```
SELECT PARSE('ABC' AS INT) AS Result
```

```
Msg 9819, Level 16, State 1, Line 1
Error converting string value 'ABC' into data type int using culture ''.
```

Since ABC cannot be converted to INT, TRY_PARSE will return NULL instead of an error

```
SELECT TRY_PARSE('ABC' AS INT) AS Result
```

Result
NULL

Example : Using TRY_PARSE() function with table data

```
SELECT Name, TRY_PARSE(Age AS INT) AS Age
FROM Employees
```

Id	Name	Age
1	Mark	40
2	John	20
3	Amy	THIRTY
4	Ben	21
5	Sara	FIFTY
6	David	25

Name	Age
Mark	40
John	20
Amy	NULL
Ben	21
Sara	NULL
David	25

If you use PARSE instead of TRY_PARSE, the query fails with an error

```
SELECT NAME, PARSE(Age AS INT) AS Age
FROM Employees
```

```
Msg 9819, Level 16, State 1, Line 1
Error converting string value 'THIRTY' into data type int using culture ''.
```

TRY CONVERT function

- TRY_CONVERT function
- Difference between CONVERT and TRY_CONVERT functions
- Difference between TRY_PARSE and TRY_CONVERT functions

TRY_CONVERT function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

TRY_CONVERT function

- Introduced in SQL Server 2012
- Converts a value to the specified data type
- Returns NULL if the provided value cannot be converted to the specified data type
- If you request a conversion that is explicitly not permitted, then TRY_CONVERT fails with an error

Syntax :

```
TRY_CONVERT ( data_type, value, [style] )
```

Style parameter is optional. The range of acceptable values is determined by the target data_type
https://msdn.microsoft.com/en-us/library/ms187928.aspx

Example : As the string can be converted to INT, the result will be 99

```
SELECT TRY_CONVERT(INT, '99') AS Result
```

Result
99

Example : As the string cannot be converted to INT, TRY_CONVERT returns NULL

```
SELECT TRY_CONVERT(INT, 'ABC') AS Result
```

Result
NULL

Example : Converting an integer to XML is not explicitly permitted. so in this case TRY_CONVERT fails with an error

```
SELECT TRY_CONVERT(XML, 10) AS Result
```

CASE statement or IIF function can be used to provide a meaningful error message instead of NULL when the conversion fails

Using CASE statement to provide a meaningful error message when the conversion fails

```
SELECT
CASE WHEN TRY_CONVERT(INT, 'ABC') IS NULL
     THEN 'Conversion Failed'
     ELSE 'Conversion Successful'
END AS Result
```

Result
Conversion Failed

Using IIF function to provide a meaningful error message when the conversion fails

```
SELECT IIF(TRY_CONVERT(INT, 'ABC') IS NULL, 'Conversion Failed',
           'Conversion Successful') AS Result
```

Result
Conversion Failed

TRY_CONVERT function in SQL Server

What is the difference between CONVERT and TRY_CONVERT

CONVERT will result in an error if the conversion fails, where as TRY_CONVERT will return NULL instead of an error

Since ABC cannot be converted to INT, CONVERT will return an error

```
SELECT CONVERT(INT, 'ABC') AS Result
```

Since ABC cannot be converted to INT, TRY_CONVERT will return NULL instead of an error

```
SELECT TRY_CONVERT(INT, 'ABC') AS Result
```

→

Result
NULL

Example : Using TRY_CONVERT() function with table data

```
SELECT Name, TRY_CONVERT(INT, Age) AS Age
FROM Employees
```

→

Id	Name	Age
1	Mark	40
2	John	20
3	Amy	THIRTY
4	Ben	21
5	Sara	FIFTY
6	David	25

→

Name	Age
Mark	40
John	20
Amy	NULL
Ben	21
Sara	NULL
David	25

If you use CONVERT instead of TRY_CONVERT, the query fails with an error

```
SELECT NAME, CONVERT(INT, Age) AS Age
FROM Employees
```

↓

```
Conversion failed when converting the nvarchar value 'THIRTY' to data type int
```

Difference between TRY_PARSE and TRY_CONVERT functions

TRY_PARSE can only be used for converting from string to date/time or number data types where as TRY_CONVERT can be used for any general type conversions.

For example, you can use TRY_CONVERT to convert a string to XML data type, where as you can do the same using TRY_PARSE

```
SELECT TRY_CONVERT(XML, '<root><child/></root>') AS [XML]
```

→

XML
<root><child /></root>

```
SELECT TRY_PARSE('<root><child/></root>' AS XML) AS [XML]
```

↓

```
Invalid data type xml in function TRY_PARSE
```

Another difference is TRY_PARSE relies on the presence of .the .NET Framework Common Language Runtime (CLR) where as TRY_CONVERT does not.

EOMONTH function

EOMONTH built-in function was introduced in *SQL* Server 2012, and it is used to get the last day of the month.

EOMONTH function in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

EOMONTH function

- Introduced in SQL Server 2012
- Returns the last day of the month of the specified date

Syntax : `EOMONTH ( start_date [, month_to_add ] )`

Parameter	Description
start_date	The date for which to return the last day of the month
month_to_add	Optional. Number of months to add to the start_date. EOMONTH adds the specified number of months to start_date, and then returns the last day of the month for the resulting date.

Example : Returns last day of the November month

```
SELECT EOMONTH('11/20/2015') AS LastDay
```

LastDay
30/11/2015

```
SELECT EOMONTH('11/20/2015') AS LastDay
```

Results | Messages

	LastDay
1	2015-11-30

```
SELECT EOMONTH('02/20/2015') AS LastDay
```

Results | Messages

	LastDay
1	2015-02-28

```
SELECT EOMONTH('02/20/2016') AS LastDay
```

Results | Messages

	LastDay
1	2016-02-29

Using EOMONTH function with table data : Returns the last day of the month from the DateOfBirth of every employee

```
SELECT Name, DateOfBirth, EOMONTH(DateOfBirth) AS LastDay
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	11/01/1980
2	John	12/12/1981
3	Amy	21/11/1979
4	Ben	14/05/1978
5	Sara	17/03/1970
6	David	05/04/1978

Name	DateOfBirth	LastDay
Mark	11/01/1980	31/01/1980
John	12/12/1981	31/12/1981
Amy	21/11/1979	30/11/1979
Ben	14/05/1978	31/05/1978
Sara	17/03/1970	31/03/1970
David	05/04/1978	30/04/1978

Use DATEPART function to retrieve just the last day instead of the full date

```
SELECT Name, DateOfBirth, DATEPART(DD,EOMONTH(DateOfBirth)) AS LastDay
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	11/01/1980
2	John	12/12/1981
3	Amy	21/11/1979
4	Ben	14/05/1978
5	Sara	17/03/1970
6	David	05/04/1978

Name	DateOfBirth	LastDay
Mark	11/01/1980	31
John	12/12/1981	31
Amy	21/11/1979	30
Ben	14/05/1978	31
Sara	17/03/1970	31
David	05/04/1978	30

DATEFROMPARTS function

DATEFROMPARTS function

- Introduced in SQL Server 2012
- Returns a date value for the specified year, month, and day
- The data type of all the 3 parameters (year, month, and day) is integer
- If invalid argument values are specified, the function returns an error
- If any of the arguments are NULL, the function returns null

Syntax : `DATEFROMPARTS ( year, month, day )`

Example : All the arguments have valid values, so DATEFROMPARTS function returns the expected date

```
SELECT DATEFROMPARTS ( 2015, 10, 25) AS [Date]
```

Date
25/10/2015

```
DATETIMEFROMPARTS : Returns DateTime
    Syntax : DATETIMEFROMPARTS (year,month,day,hour,minute,seconds,milliseconds)

SMALLDATETIMEFROMPARTS : Returns SmallDateTime
    Syntax : SMALLDATETIMEFROMPARTS (year,month,day,hour,minute)
```

Difference between DateTime and SmallDateTime

DateTime v/s SmallDateTime

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Attribute	SmallDateTime	DateTime
Date Range	January 1, 1900, through June 6, 2079	January 1, 1753, through December 31, 9999
Time Range	00:00:00 through 23:59:59	00:00:00 through 23:59:59.997
Accuracy	1 Minute	3.33 Milli-seconds
Size	4 Bytes	8 Bytes
Default value	1900-01-01 00:00:00	1900-01-01 00:00:00

```
-- Values with 29.998 seconds or lower are rounded down
-- SmallDateTime - Time rounded to 12:35
Insert into Employees ([SmallDateTime]) values ('01/01/1995 12:35:29.998'

Select * from Employees
Truncate table Employees

-- Values with 29.999 seconds or higher are rounded up
-- SmallDateTime - Time rounded to 12:36
Insert into Employees ([SmallDateTime]) values ('01/01/1995 12:35:29.999'

Select * from Employees
Truncate table Employees
```

```
26 -- Values are rounded to increments of .000, .003, or .007 seconds
27 -- DateTime - Time rounded to 01/01/99 23:59:59.990
28 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.990')
29 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.991')
30
31 Select * from Employees
```

	SmallDateTime	DateTime
1	NULL	1995-01-01 23:59:59.990
2	NULL	1995-01-01 23:59:59.990

```
35 -- DateTime - Time rounded to 01/01/99 23:59:59.993
36 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.992')
37 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.993')
38 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.994')
39
40 Select * from Employees
41 Truncate table Employees
42
43
44 -- DateTime - Time rounded to 01/01/99 23:59:59.997
45 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.995')
46 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.996')
47 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.997')
48 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.998')
```

	SmallDateTime	DateTime
1	NULL	1995-01-01 23:59:59.993
2	NULL	1995-01-01 23:59:59.993
3	NULL	1995-01-01 23:59:59.993

```
44 -- DateTime - Time rounded to 01/01/99 23:59:59.997
45 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.995')
46 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.996')
47 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.997')
48 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.998')
49
50 Select * from Employees
51 Truncate table Employees
```

	SmallDateTime	DateTime
1	NULL	1995-01-01 23:59:59.997
2	NULL	1995-01-01 23:59:59.997
3	NULL	1995-01-01 23:59:59.997
4	NULL	1995-01-01 23:59:59.997

```
55 Insert into Employees ([DateTime]) values ('01/01/1995 23:59:59.999')
56
57 Select * from Employees
58 Truncate table Employees
```

	SmallDateTime	DateTime
1	NULL	1995-01-02 00:00:00.000

DateTime2FromParts function

DateTime2FromParts function

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

DateTime2FromParts function

- Introduced in SQL Server 2012
- Returns DateTime2
- The data type of all the parameters is integer
- If invalid argument values are specified, the function returns an error
- If any of the required arguments are NULL, the function returns null
- If the precision argument is null, the function returns an error

Syntax : DATETIME2FROMPARTS(year,month,day,hour,minute,seconds,fractions,precision)

Example : All the function arguments have valid values, so DATETIME2FROMPARTS returns DATETIME2 value as expected

```
SELECT DATETIME2FROMPARTS (2015, 11, 15, 20, 55, 55, 0, 0) AS [DateTime2]
```

DateTime2
2015-11-15 20:55:55

```
SELECT DATETIME2FROMPARTS ( 2015, 11, 15, 20, 55, 55, 5, 7) AS [DateTime2]
```

Results | Messages

	DateTime2
1	2015-11-15 20:55:55.0000005

DateTime2FromParts function

Example : Invalid value specified for month parameter, so the function returns an error

```
SELECT DATETIME2FROMPARTS ( 2015, 15, 15, 20, 55, 55, 0, 0 ) AS [DateTime2]
```

```
Cannot construct data type datetime2, some of the
arguments have values which are not valid
```

Example : If any of the required arguments are NULL, the function returns null. NULL specified for month parameter, so the function returns NULL

```
SELECT DATETIME2FROMPARTS ( 2015, NULL, 15, 20, 55, 55, 0, 0 ) AS [DateTime2]
```

DateTime2
NULL

Example : If the precision argument is null, the function returns an error

```
SELECT DATETIME2FROMPARTS ( 2015, 15, 15, 20, 55, 55, 0, NULL ) AS [DateTime2]
```

```
Scale argument is not valid. Valid expressions for data
type datetime2 scale argument are integer constants and
integer constant expressions
```

Difference between DateTime and DateTime2

DateTime v/s DateTime2

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Differences between DateTime and DateTime2

Attribute	DateTime	DateTime2
Date Range	January 1, 1753, through December 31, 9999	January 1, 0001, through December 31, 9999
Time Range	00:00:00 through 23:59:59.997	00:00:00 through 23:59:59.9999999
Accuracy	3.33 Milli-seconds	100 nanoseconds
Size	8 Bytes	6 to 8 Bytes (Depends on the precision)
Default Value	1900-01-01 00:00:00	1900-01-01 00:00:00

DATETIME2 has a bigger date range than DATETIME. Also, DATETIME2 is more accurate than DATETIME

So I would recommend using DATETIME2 over DATETIME when possible

I think the only reason for using DATETIME over DATETIME2 is for backward compatibility

Notice as the precision increases the storage size also increases

Precision	DateValue	StorageSize
Precision - 0	2015-10-20 15:09:12.0000000	6
Precision - 1	2015-10-20 15:09:12.1000000	6
Precision - 2	2015-10-20 15:09:12.1200000	6
Precision - 3	2015-10-20 15:09:12.1230000	7
Precision - 4	2015-10-20 15:09:12.1235000	7
Precision - 5	2015-10-20 15:09:12.1234600	8
Precision - 6	2015-10-20 15:09:12.1234570	8
Precision - 7	2015-10-20 15:09:12.1234567	8

```
Declare @TempTable Table
(
    DateTime2Precision0 DateTime2(0),
    DateTime2Precision1 DateTime2(1),
    DateTime2Precision2 DATETIME2(2),
    DateTime2Precision3 DATETIME2(3),
    DateTime2Precision4 DATETIME2(4),
    DateTime2Precision5 DATETIME2(5),
    DateTime2Precision6 DATETIME2(6),
    DateTime2Precision7 DATETIME2(7)
)
```

```
Insert into @TempTable values
(
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567',
    '2015-10-20 15:09:12.1234567'
)
```

```
Select 'Precision - 0' AS [Precision],
        DateTime2Precision0 AS [DateTime],
        DATALENGTH(DateTime2Precision0) AS [Storage Size]
From @TempTable

Union All

Select 'Precision - 1' AS [Precision],
        DateTime2Precision1 AS [DateTime],
        DATALENGTH(DateTime2Precision1) AS [Storage Size]
From @TempTable

Union All

Select 'Precision - 2' AS [Precision],
        DateTime2Precision2 AS [DateTime],
        DATALENGTH(DateTime2Precision) AS [Storage Size]
From @TempTable

UNION ALL
```

Results | Messages

	Precision	DateTime	Storage Size
1	Precision - 0	2015-10-20 15:09:12.0000000	6
2	Precision - 1	2015-10-20 15:09:12.1000000	6
3	Precision - 2	2015-10-20 15:09:12.1200000	6
4	Precision - 3	2015-10-20 15:09:12.1230000	7
5	Precision - 4	2015-10-20 15:09:12.1235000	7
6	Precision - 5	2015-10-20 15:09:12.1234600	8
7	Precision - 6	2015-10-20 15:09:12.1234570	8
8	Precision - 7	2015-10-20 15:09:12.1234567	8

CREATING A TABLE:

```
Create table tblProducts
(
    Id int primary key identity,
    Name nvarchar(25),
    [Description] nvarchar(50),
    Price int
)
Go
```

ENTERING VALUES IN TO TABLE USING WHILE LOOP:

```
-- Populate tblProducts
Declare @Start int
Set @Start = 1

Declare @Name varchar(25)
Declare @Description varchar(50)

While(@Start <= 100)
Begin
    Set @Name = 'Product - ' + LTRIM(@Start)
    Set @Description = 'Product Description - ' + LTRIM(@Start)
    Insert into tblProducts values (@Name, @Description, @Start * 10)
    Set @Start = @Start + 1
End
```

OFFSET SKIPS THE NUMBER OF ROWS AND FETCHES THE NEXT SET OF GIVEN NUMBER OF ROWS. BELOW WE GET ROWS FROM 11TH ROW 20TH ROW AS THE FIRST 10TH ROWS ARE SKIPPED.

```
SELECT * FROM tblProducts
ORDER BY Id
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY
```

Results | Messages

	Id	Name	Description	Price
1	11	Product - 11	Product Description - 11	110
2	12	Product - 12	Product Description - 12	120
3	13	Product - 13	Product Description - 13	130
4	14	Product - 14	Product Description - 14	140
5	15	Product - 15	Product Description - 15	150
6	16	Product - 16	Product Description - 16	160
7	17	Product - 17	Product Description - 17	170
8	18	Product - 18	Product Description - 18	180
9	19	Product - 19	Product Description - 19	190
10	20	Product - 20	Product Description - 20	200

```
Create proc spGetRowsByPageNumberAndPageSize
@PageNumber INT,
@PageSize INT
AS
BEGIN
    SELECT * FROM tblProducts
    ORDER BY Id
    OFFSET (@PageNumber - 1) * @PageSize ROWS
    FETCH NEXT @PageSize ROWS ONLY
END
```

IF WE GIVE @PAGENUMBER VALUE AS 1 AND PAGESIZE AS 10 THEN WE GET PAGENUMBERS FROM 1 TO 10.

IF WE GIVE @PAGENUMBER VALUE AS 10 AND PAGESIZE AS 10, THEN WE GET ROWS FROM 10 TO 20.

```
CREATE PROCEDURE spGetRowsByPageNumberAndSize
@PageNumber INT,
@PageSize INT
AS
BEGIN
    SELECT * FROM tblProducts
    ORDER BY Id
    OFFSET (@PageNumber - 1) * @PageSize ROWS
    FETCH NEXT @PageSize ROWS ONLY
END
```

```
EXECUTE spGetRowsByPageNumberAndSize 3, 10
```

Id	Name	Description	Price
21	Product - 21	Product Description - 21	210
22	Product - 22	Product Description - 22	220
23	Product - 23	Product Description - 23	230
24	Product - 24	Product Description - 24	240
25	Product - 25	Product Description - 25	250
26	Product - 26	Product Description - 26	260
27	Product - 27	Product Description - 27	270
28	Product - 28	Product Description - 28	280
29	Product - 29	Product Description - 29	290
30	Product - 30	Product Description - 30	300

IDENTIFYING OBJECT DEPENDENCIES IN SQL SERVER:

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

Create table Employees
(
    Id int primary key identity,
    Name nvarchar(50),
    Gender nvarchar(10),
    DeptId int foreign key references Departments(Id)
)
Go
```

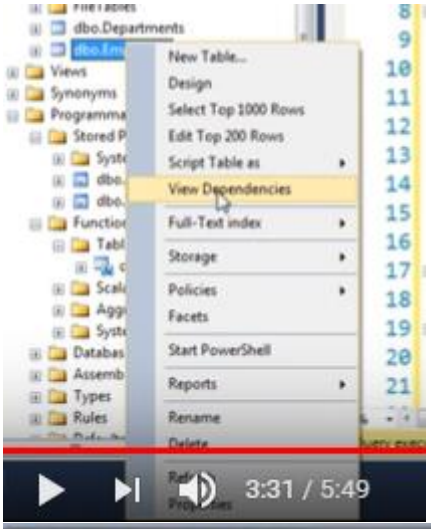
```
Create procedure sp_GetEmployees
as
Begin
    Select * from Employees
End
Go

Create procedure sp_GetEmployeesandDepartments
as
Begin
    Select Employees.Name as EmployeeName, Departments.Name as DepartmentName
    from Employees
    join Departments
    on Employees.DeptId = Departments.Id
End
Go
```

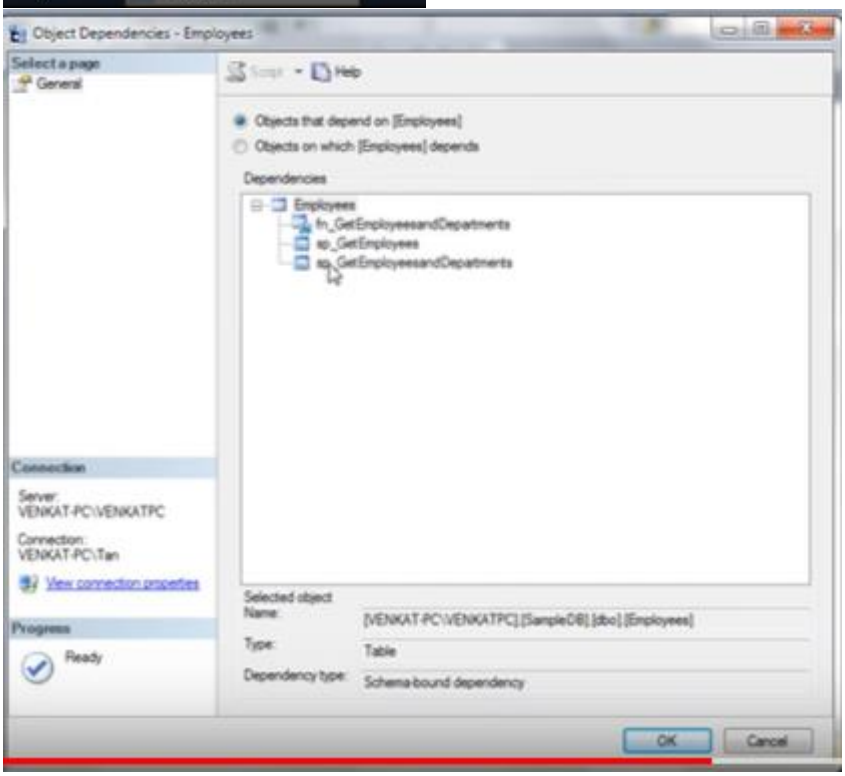
```
Create view VwDepartments
as
Select * from Departments
Go

Create function fn_GetEmployeesandDepartments()
returns table
as
Return
      Select Employees.Name as EmployeeName, Departments.Name as DepartmentName
      from Employees
      join Departments
      on Employees.DeptId = Departments.Id
```

WILL NOT BE ABLE TO DELETE DEPARTMENT TABLE AS THE EMPLOYEE TABLE HAS A FOREIGN KEY CONSTRAINT REFERENCING THE PRIMARY KEY IN DEPARTMENT TABLE.

HOWEVER, WE CAN STILL DELETE THE EMPLOYEE TABLE. ALL THE STORED PROCEDURES AND FUNCTIONS THAT DEPENDS UPON EMPLOYEE TABLE WILL LOSE THEIR FUNCTIONALITY. THEREFORE, WE MUST BE CAREFUL BEFORE DELETING ANY DATABASE OBJECTS TO MAKE SURE THAT IT DOES NOT HAVE ANY DEPENDENCIES.

HOW TO CHECK OBJECT DEPENDENCIES: RIGHT CLICK ON THE TABLE AND ->VIEW DEPENDENCIES





Identifying object dependencies is important especially when you intend to modify or delete an object upon which other objects depend. Otherwise you may risk breaking the functionality

sys dm sql referencing entities

Returns all the objects that depend on Employees table

```
Select * from sys.dm_sql_referencing_entities('dbo.Employees','Object')
```

Difference between referencing entity and referenced entity

A dependency is created between two objects when one object appears by name inside a SQL statement stored in another object. The object which is appearing inside the SQL expression is known as REFERENCED ENTITY and the object which has the SQL expression is known as a REFERENCING ENTITY

Referencing entities	sys.dm_sql_referencing_entities
Referenced entities	sys.dm_sql_referenced_entities

```
Create view VwEmployees
as
Select * from Employees
```

→ REFERENCING ENTITY

→ REFERENCED ENTITY

REFERENCING_ENTITIES RETURNS ALL THE OBJECTS THAT DEPENDS UPON A PARTICULAR OBJECT OR TABLE

```
Select * from sys.dm_sql_referencing_entities('dbo.Employees','Object')

Select * from sys.dm_sql_referenced_entities
('dbo.sp_GetEmployeesandDepartments','Object')
```

	referencing_schema_name	referencing_entity_name	referencing_id	referencing_class	referencing_class_desc	is_caller_dependent
1	dbo	fn_GetEmployeesandDepartments	1717581157	1	OBJECT_OR_COLUMN	0
2	dbo	sp_GetEmployees	1733581214	1	OBJECT_OR_COLUMN	0
3	dbo	sp_GetEmployeesandDepartments	1685581043	1	OBJECT_OR_COLUMN	0

REFERENCED_ENTITIES RETURNS ALL THE OBJECTS THAT THE PARTICULAR OBJECT DEPENDS UPON. SUCH ALL THE OBJECT THAT STORED_PROC BELOW DEPENDS UPON:

```
Select * from sys.dm_sql_referenced_entities
('dbo.sp_GetEmployeesandDepartments','Object')
```

	referencing_minor_id	referenced_server_name	referenced_database_name	referenced_schema_name	referenced_entity_name	referenced_minor_name	referenced_id	ref
1	0	NULL	NULL	NULL	Employees	NULL	226099846	0
2	0	NULL	NULL	NULL	Employees	Name	226099846	2
3	0	NULL	NULL	NULL	Employees	DeptId	226099846	4
4	0	NULL	NULL	NULL	Departments	NULL	1589580701	0
5	0	NULL	NULL	NULL	Departments	Id	1589580701	1
6	0	NULL	NULL	NULL	Departments	Name	1589580701	2

The following query returns all the REFERENCED ENTITIES of the stored procedure sp_GetEmployeesandDepartments

```
Select * from sys.dm_sql_referenced_entities('dbo.sp_GetEmployeesandDepartments','Object')
```

Please note : For both these dynamic management functions to work we need to specify the schema name as well

Difference between Schema-bound dependency and Non-schema-bound dependency

Schema-bound dependency : Schema-bound dependency prevents referenced objects from being dropped or modified as long as the referencing object exists

Example : A view created with SCHEMABINDING, or a table created with foreign key constraint.

Non-schema-bound dependency : A non-schema-bound dependency doesn't prevent the referenced object from being dropped or modified.

CAN NOT DROP EMPLOYEES TABLE IF IT IS SCHEMABOUND IN THE VIEW TABLE

SQLQuery2.sql - V...ENKAT-PC\Tan (53))* | SQLQuery1.sql - V...ENKAT-PC\Tan (52))*

```
Create view VwEmployees
WITH SCHEMABINDING
as
Select Id, Name, Gender, DeptId from dbo.Employees
```

sp_depends

1. A system stored procedure that returns object dependencies
2. For example,
   1. If you specify a table name as the argument, then the views and procedures that depend on the specified table are displayed
   2. If you specify a view or a procedure name as the argument, then the tables and views on which the specified view or procedure depends are displayed

Syntax: `Execute sp_depends 'ObjectName'`

Returns the stored procedure that depends on table Employees

```
sp_depends 'Employees'
```

name	type
dbo.sp_GetEmployees	stored procedure

Returns the name of the table and the respective column names on which the stored procedure sp_GetEmployees depends

```
sp_depends 'sp_GetEmployees'
```

name	type	updated	selected	column
dbo.Employees	user table	no	yes	Id
dbo.Employees	user table	no	yes	Name
dbo.Employees	user table	no	yes	Gender

sp_depends does not report dependencies correctly, if the table on which the stored procedure depends is deleted and recreated.

sp_depends is on the deprecation path. This might be removed from the future versions of SQL server.

Sequence object

Sequence object in SQL Server 2012

```
CREATE SEQUENCE [schema_name . ] sequence_name
    [ AS [ built_in_integer_type | user-defined_integer_type ] ]
    [ START WITH <constant> ]
    [ INCREMENT BY <constant> ]
    [ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
    [ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
    [ CYCLE | { NO CYCLE } ]
    [ { CACHE [ <constant> ] } | { NO CACHE } ]
    [ ; ]
```

Property	Description
DataType	Built-in integer type (tinyint , smallint, int, bigint, decimal etc...) or user-defined integer type. Default bigint.
START WITH	The first value returned by the sequence object
INCREMENT BY	The value to increment or decrement by. The value will be decremented if a negative value is specified.
MINVALUE	Minimum value for the sequence object
MAXVALUE	Maximum value for the sequence object
CYCLE	Specifies whether the sequence object should restart when the max value (for incrementing sequence object) or min value (for decrementing sequence object) is reached. Default is NO CYCLE, which throws an error when minimum or maximum value is exceeded.
CACHE	Cache sequence values for performance. Default value is CACHE.

Sequence object in SQL Server 2012

Creating an Incrementing Sequence

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 1
INCREMENT BY 1
```

Generating the Next Sequence Value

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

Retrieving the current sequence value

```
SELECT current_value FROM sys.sequences WHERE name = 'SequenceObject'
```

Reset the sequence value

```
ALTER SEQUENCE [SequenceObject] RESTART WITH 1
```

Using sequence value in an INSERT query

```
INSERT INTO Employees VALUES (NEXT VALUE for [dbo].[SequenceObject], 'Ben', 'Male')
INSERT INTO Employees VALUES (NEXT VALUE for [dbo].[SequenceObject], 'Sara', 'Female')
```

Creating a decrementing Sequence :

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 100
INCREMENT BY -1
```

Specifying MIN and MAX values for the sequence

```
CREATE  SEQUENCE [dbo].[SequenceObject]
        START WITH 100
        INCREMENT BY 10
        MINVALUE 100
        MAXVALUE 150
```

If you call NEXT VALUE, when the value reaches 150 (MAXVALUE)

```
The sequence object 'SequenceObject' has reached its minimum or maximum value.
Restart the sequence object to allow new values to be generated.
```

Sequence object in SQL Server 2012

Recycling Sequence values : When the sequence object has reached it's maximum value, and if you want to restart from the minimum value, set CYCLE option

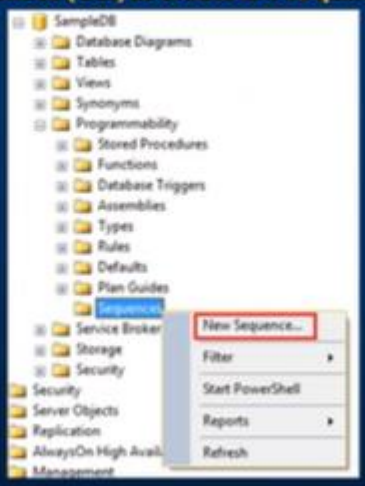
```
ALTER   SEQUENCE [dbo].[SequenceObject]
        INCREMENT BY 10
        MINVALUE 100
        MAXVALUE 150
        CYCLE
```

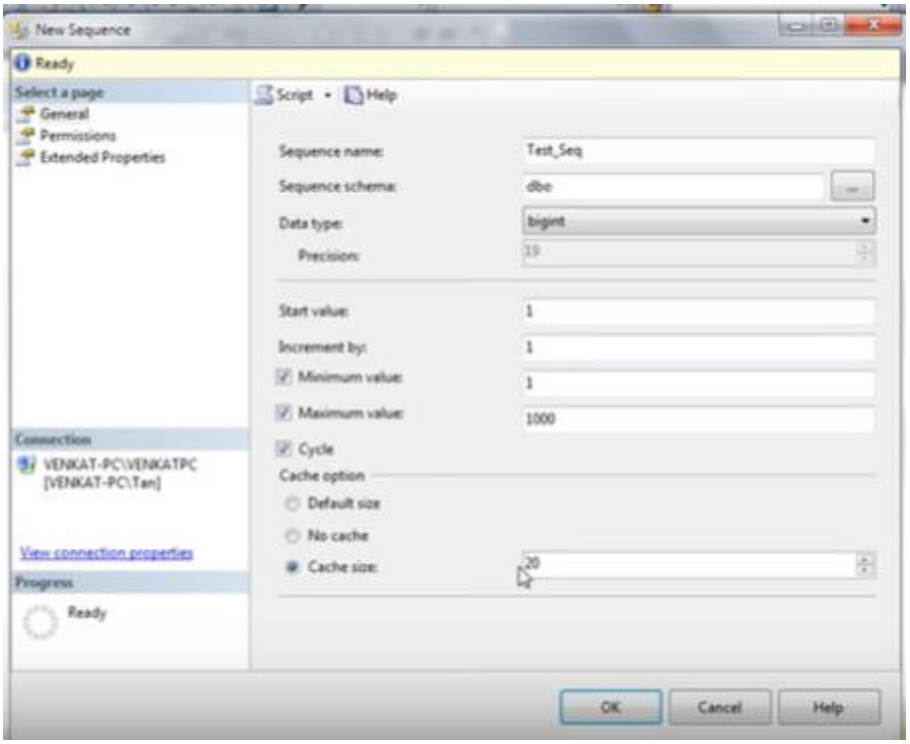
To improve performance, the Sequence object values can be cached using the CACHE option.

```
CREATE  SEQUENCE [dbo].[SequenceObject]
        START WITH 1
        INCREMENT BY 1
        CACHE 10
```

Using SQL Server Graphical User Interface (GUI) to create the sequence object :

1. Expand the database folder
2. Expand Programmability folder
3. Right click on Sequences folder
4. Select New Sequence





Difference between sequence and identity

Difference between Sequence & Identity

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Identity	Sequence
Identity property is a table column property meaning it is tied to the table	Sequence is a user-defined database object and is not tied to any specific table meaning it's value can be shared by multiple tables.
To generate the next identity value, a row has to be insetred into the table	With sequence object there is no need to insert a row into the table to generate the next sequence value. You can use NEXT VALUE FOR clause to generate the next sequence value.
Maximum value for the identity property cannot be specified. The maximum value will be the maximum value of the correspoding column data type.	With the sequence object you can use the MAXVALUE option to specify the maximum value. If the MAXVALUE option is not specified for the sequence object, then the maximum value will be the maximum value of it's data type.
Identity property does not have any option to automatically restart the identity values	With the Sequence object CYCLE option can be used to specify whether the sequence should restart automatically when the max value (for incrementing sequence object) or min value (for decrementing sequence object) is reached.

Guid in SQL Server

Guid in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is a GUID in SQL Server

A GUID is a 16 byte binary data type that is globally unique. GUID stands for Global Unique Identifier. The terms GUID and UNIQUEIDENTIFIER are used interchangeably

```
Declare @ID UNIQUEIDENTIFIER
```

How to create a GUID

To create a GUID in SQL Server use NEWID() function

For example, SELECT NEWID() creates a GUID that is guaranteed to be unique across tables, databases, and servers.

Example GUID : 215D55D8-1683-42D8-927C-B151DBE4EE0B

```
Declare @ID UNIQUEIDENTIFIER
SET @ID = NEWID()
SELECT @ID
```

100 %

Results | Messages

	(No column name)
1	10479929-CF45-4DF1-AEEB-58EB95B3DC53

```
Create Database USADB
Go

Use USADB
Go

Create Table USACustomers1
(
    ID uniqueidentifier primary key default newid(),
    Name nvarchar(50)
)
Go

Insert Into USACustomers1 Values (default, 'Tom')
Insert Into USACustomers1 Values (default, 'Mike')

Select * From USADB.dbo.USACustomers1
```

WHEN WE USE UNIQUEIDENTIFIER DATATYPE FOR CUSTOMERS1, INDIACUSTOMERS1 AND USCUSTOMERS1 TABLE ID COLUMN, THEN WE WILL GET UNIQUE VALUES FOR ID COLUMN THROUGHOUT DIFFERENT TABLES IN THE DATABASE AND ACROSS THE SERVER.

```
Use IndiaDB
Go

Create Table Customers1
(
    ID uniqueidentifier primary key,
    Name nvarchar(50)
)
Go

Insert Into Customers1
Select * from IndiaDB.dbo.IndiaCustomers1
Union All
Select * from USADB.dbo.USACustomers1
```

Guid in SQL Server

Advantages :

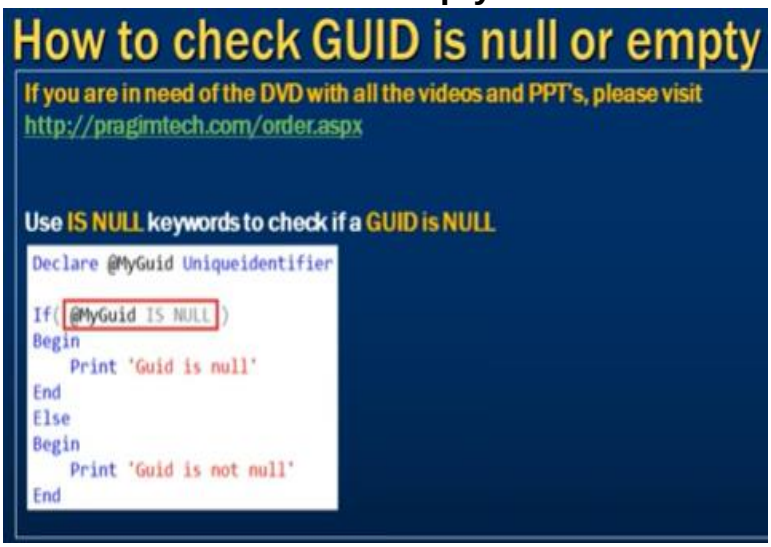
- A GUID is unique across tables, databases and servers
- Useful if you're consolidating records from multiple SQL Servers into a single table

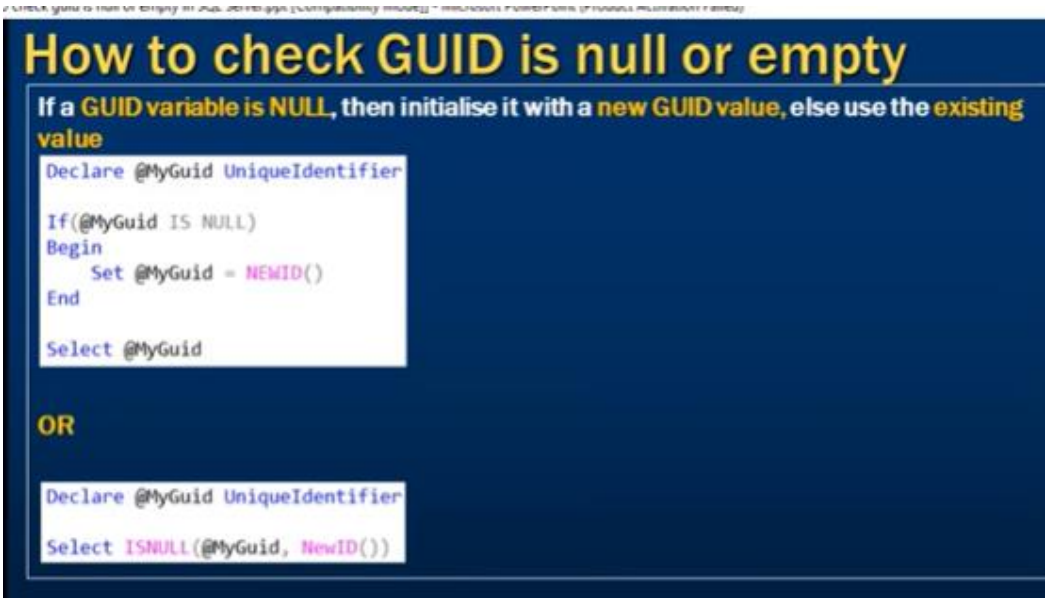
Disadvantages :

- Size is 16 bytes, where as INT is only 4 bytes
- One of the largest datatypes in SQL Server
- An Index built on a GUID is larger and slower
- Hard to read compared to INT

Summary : Only use a GUID when you really need a globally unique identifier. In all other cases it is better to use an INT data type

How to check GUID is null or empty





How to check GUID is null or empty

What is an empty GUID : An empty GUID is a GUID with all ZEROS

00000000-0000-0000-0000-000000000000

To create an EMPTY GUID

```
SELECT CAST(CAST(0 AS BINARY) AS UNIQUEIDENTIFIER)
OR
SELECT CAST(0x0 AS UNIQUEIDENTIFIER)
```

Two options to check if a GUID is an empty GUID

Option 1 :
```
If(@MyGuid = '00000000-0000-0000-0000-000000000000')
```

Option 2 :
```
If(@MyGuid = Cast(0x0 as Uniqueidentifier))
```

Dynamic SQL

Dynamic SQL in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is Dynamic SQL

Dynamic SQL is a SQL built from strings at runtime

Employee Search Form

First Name	
Last Name	
Gender	
Salary	

Search Clear

```
Create Procedure spSearchEmployees
@FirstName nvarchar(100),
@LastName nvarchar(100),
@Gender nvarchar(50),
@Salary int
As
Begin

    Select * from Employees where
    (FirstName = @FirstName OR @FirstName IS NULL) AND
    (LastName  = @LastName  OR @LastName  IS NULL) AND
    (Gender    = @Gender    OR @Gender    IS NULL) AND
    (Salary    = @Salary    OR @Salary    IS NULL)
End
```

We have a search form as shown above and we want to search the following employees table and the stored procedure for this search is shown on right.

	ID	FirstName	LastName	Gender	Salary
1	1	Mark	Hastings	Male	60000
2	2	Steve	Pound	Male	45000
3	3	Ben	Hoskins	Male	70000
4	4	Philip	Hastings	Male	45000
5	5	Mary	Lambeth	Female	30000
6	6	Valarie	Vikings	Female	35000
7	7	John	Stanmore	Male	80000

Notice that the search form has 4 search fields and the stored procedure has 4 parameters corresponding to those search fields. And within the body of the stored procedure we are retrieving all the rows and columns.

When we execute this stored procedure to search an employee whose first name is Mark and Last Name is Hastings then we get:

```
Execute spSearchEmployees 'Mark','Hastings', NULL, NULL
```

Results | Messages

	ID	FirstName	LastName	Gender	Salary
1	1	Mark	Hastings	Male	60000

Dynamic SQL in SQL Server

```
Declare @sql nvarchar(1000)
Declare @params nvarchar(1000)

Set @sql = 'Select * from Employees where FirstName=@FirstName and LastName=@LastName'
Set @params = '@FirstName nvarchar(100), @LastName nvarchar(100)'

Execute sp_executesql @sql, @params, @FirstName='Ben',@LastName='Hoskins'
```

To execute the Dynamic SQL use system stored procedure sp_executesql. It takes two pre-defined parameters and any number of user-defined parameters

Parameter	Description
@statement	The is the first parameter which is mandatory, and contains the SQL statements to execute
@params	This is the second parameter and is optional. This is used to declare parameters specified in @statement

The rest of the parameters are the parameters that you declared in @params, and you pass them the same way as you pass parameters to a stored procedure

```
Declare @sql nvarchar(1000)
Declare @params nvarchar(1000)

Set @sql = 'Select * from Employees' + ' Where FirstName=@FirstName and LastName=@LastName'
Set @params = '@FirstName nvarchar(100), @LastName nvarchar(100)'

--Print @sql

Execute sp_executesql @sql, @params, @LastName='Hastings', @FirstName='Mark'
```

100 %

Results | Messages

	ID	FirstName	LastName	Gender	Salary
1	1	Mark	Hastings	Male	60000