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

--120-- REVERSE PIVOT TABLE IN SQL SERVER
--121-- CHOOSE FUNCTION IN SQL SERVER
--122-- IIF FUNCTION IN SQL SERVER
--123-- TRY_PARSE FUNCTION IN SQL SERVER 2012
--124-- TRY_CONVERT FUNCTION IN SQL SERVER 2012
--125-- EOMONTH FUNCTION IN SQL SERVER 2012
--126-- DATEFROMPARTS FUNCTION
--127- DIFFERENCE BETWEEN DATETIME AND SAMLLDATETIME IN SQL SERVER
--128-- DATETIME2FROMPARTS FUNCTION IN SQL SERVER 2012
--129-- DIFFERENCE BETWEEN DATETIME AND DATETIME2 IN SQL SERVER
--130-- OFFSET FETCH NEXT IN SQL SERVER 2012
--131-- IDENTIFYING OBJECT DEPENDENCIES IN SQL SERVER
--132-- SYS.DM_SQL_REFERENCING_ENTITIES IN SQL SERVER
--133-- SP_DEPENDS IN SQL SERVER
--134-- SEQUENCE OBJECT IN SQL SERVER 2012
--135-- DIFFERENCE BETWEEN SEQUENCE AND IDENTITY IN SQL SERVER
--136-- GUID IN SQL SERVER
--137-- HOW TO CHECK GUID IS NULL OR EMPTY IN SQL SERVER
--138-- DYNAMIC SQL IN SQL SERVER
--139-- IMPLEMENT SEARCH WEB PAGE USING ASP.NET AND STORED PROCEDURE
--140-- IMPLEMENT SEARCH WEB PAGE USING ASP.NET AND DYNAMIC SQL
--141-- PREVENT SQL INJECTIO WITH DYNAMIC SQL
--142-- SQL SERVER QUERY PLAN CACHE
--143-- EXEC VS SP_EXECUTESQL IN SQL SERVER
--144-- DYNAMIC SQL TABLE NAME VARIABLE
--145-- QUOTENAME FUNCTION IN SQL SERVER
--146-- DYNAMIC SQL VS STORED PROCEDURE
--147-- DYNAMIC SQL OUTPUT PARAMETER
--148-- TEMP TABLES IN DYNAMIC SQL

--120-- REVERSE PIVOT TABLE IN SQL SERVER

In this video we will discuss if its always possible to reverse what PIVOT operator has done using UNPIVOT operator.

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.

Let us understand this with an example. We will use the following table tblProductSales for the examples in this video.
reverse pivot table sql

SQL Script to create tblProductSales table

```
Create Table tblProductSales
(
    SalesAgent nvarchar(10),
    Country nvarchar(10),
    SalesAmount int
)
Go

Insert into tblProductSales values('David','India',960)
Insert into tblProductSales values('David','US',520)
Insert into tblProductSales values('John','India',970)
Insert into tblProductSales values('John','US',540)
Go
```

Lets now use the PIVOT operator to turn ROWS into COLUMNS

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

The above query produces the following output

convert rows to columns sql

Now lets use the UNPIVOT operator to reverse what PIVOT operator has done

```
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
```

The above query reverses what PIVOT operator has done, and we get the original data back as shown below. 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.

pivot and unpivot table sql server

Now execute the following INSERT statement to insert a new row into tblProductSales table.
Insert into tblProductSales values('David','India',100)

With this new row in the table, if you execute the following PIVOT query data will be aggregated

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

The following is the result of the above query
pivot and unpivot examples in sql server

Now if we use UNPIVOT opertaor with the above query, we wouldnt get our orginial data back as the PIVOT operator has already aggrgated the data, and there is no way for SQL Server to know how to undo the aggregations.

```
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
```

Notice that for SalesAgent - David and Country - India we get only one row.
In the original table we had 2 rows for the same combination.
undo pivot table sql server

--121-- CHOOSE FUNCTION IN SQL SERVER

Choose function
Introduced in SQL Server 2012
Returns the item at the specified index from the list of available values
The index position starts at 1 and NOT 0 (ZERO)
Syntax : CHOOSE(index, val_1, val_2, ...)

Example : Returns the item at index position 2

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

Output :
Choose function in SQL Server

Example : Using CHOOSE() function with table data.
We will use the following Employees table for this example.
sql server 2012 choose function

SQL Script to create Employees table

```
Create table Employees
(
    Id int primary key identity,
    Name nvarchar(10),
    DateOfBirth date
)
Go

Insert into Employees values ('Mark', '01/11/1980')
Insert into Employees values ('John', '12/12/1981')
Insert into Employees values ('Amy', '11/21/1979')
Insert into Employees values ('Ben', '05/14/1978')
Insert into Employees values ('Sara', '03/17/1970')
Insert into Employees values ('David', '04/05/1978')
Go
```

We want to display Month name along with employee Name and Date of Birth.
choose function in sql 2012

Using CASE statement in SQL Server

```
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
```

Using CHOOSE function in SQL Server : The amount of code we have to write is lot less than using CASE statement.

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

--122-- IIF FUNCTION IN SQL SERVER

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
```

Output :
iif function in sql server example

Example : Using IIF() function with table data.
We will use the following Employees table for this example.
sql server iif function example

SQL Script to create Employees table

```
Create table Employees
(
    Id int primary key identity,
    Name nvarchar(10),
    GenderId int
)
Go

Insert into Employees values ('Mark', 1)
Insert into Employees values ('John', 1)
Insert into Employees values ('Amy', 2)
Insert into Employees values ('Ben', 1)
Insert into Employees values ('Sara', 2)
Insert into Employees values ('David', 1)
Go
```

Write a query to display Gender along with employee Name and GenderId.
We can achieve this either by using CASE or IIF.

iif function in sql server 2012

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
```

--123-- TRY_PARSE FUNCTION IN SQL SERVER 2012

In this video we will discuss
TRY_PARSE function
Difference between PARSE and TRY_PARSE functions

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 : Convert string to INT. As the string can be converted to INT,
the result will be 99 as expected.

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

Output :
try_parse function in sql server 2012

Example : Convert string to INT. The string cannot be converted to INT, so TRY_PARSE returns NULL

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

Output :
sql server tryparse

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
```

Output : As the conversion fails, you will now get a message 'Conversion Failed' instead of NULL
sql server try_parse

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
```

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
```

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

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

Example : Using TRY_PARSE() function with table data.
We will use the following Employees table for this example.
try_parse in sql server 2012

SQL Script to create Employees table

```
Create table Employees
(
    Id int primary key identity,
    Name nvarchar(10),
    Age nvarchar(10)
)
Go
```

```
Insert into Employees values ('Mark', '40')
Insert into Employees values ('John', '20')
Insert into Employees values ('Amy', 'THIRTY')
Insert into Employees values ('Ben', '21')
Insert into Employees values ('Sara', 'FIFTY')
Insert into Employees values ('David', '25')
Go
```

The data type of Age column is nvarchar. So string values like (THIRTY, FIFTY) are also stored.
Now, we want to write a query to convert the values in Age column to int and return along with the Employee name.
Notice TRY_PARSE function returns NULL for the rows where age cannot be converted to INT.

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

try parse in sql server

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

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

The above query returns the following error
Error converting string value 'THIRTY' into data type int using culture

--124-- TRY_CONVERT FUNCTION IN SQL SERVER 2012

In this video we will discuss
TRY_CONVERT function
Difference between CONVERT and TRY_CONVERT functions
Difference between TRY_PARSE and TRY_CONVERT functions

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.
For the list of all possible values for style parameter, please visit the following MSDN article
https://msdn.microsoft.com/en-us/library/ms187928.aspx

Example : Convert string to INT. As the string can be converted to INT, the result will be 99 as expected.

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

Output :
try_convert function in sql server 2012

Example : Convert string to INT. The string cannot be converted to INT, so TRY_CONVERT returns NULL

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

Output :

try convert function in sql

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
```

If you want to provide a meaningful error message instead of NULL when the conversion fails,
you can do so using CASE statement or IIF function.

Example : 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
```

Output : As the conversion fails, you will now get a message 'Conversion Failed' instead of NULL
sql server try_convert

Example : 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
```

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
```

Example : Using TRY_CONVERT() function with table data.
We will use the following Employees table for this example.
try_convert in sql server 2012

SQL Script to create Employees table

```
Create table Employees
(
```

```
    Id int primary key identity,
    Name nvarchar(10),
    Age nvarchar(10)
)
Go

Insert into Employees values ('Mark', '40')
Insert into Employees values ('John', '20')
Insert into Employees values ('Amy', 'THIRTY')
Insert into Employees values ('Ben', '21')
Insert into Employees values ('Sara', 'FIFTY')
Insert into Employees values ('David', '25')
Go
```

The data type of Age column is nvarchar. So string values like (THIRTY, FIFTY) are also stored. Now, we want to write a query to convert the values in Age column to int and return along with the Employee name.
Notice TRY_CONVERT function returns NULL for the rows where age cannot be converted to INT.

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

try convert sql

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

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

The above query returns the following error
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

Converting a string to XML data type using TRY_CONVERT

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

The above query produces the following
try_parse vs try_convert sql server

Converting a string to XML data type using TRY_PARSE

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

The above query will result in the following error
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.

--125-- EOMONTH FUNCTION IN SQL SERVER 2012

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])

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 month November
SELECT EOMONTH('11/20/2015') AS LastDay

Output :
sql eomonth example

Example : Returns last day of the month of February from a NON-LEAP year
SELECT EOMONTH('2/20/2015') AS LastDay

Output :
eomonth function in sql server 2012

Example : Returns last day of the month of February from a LEAP year
SELECT EOMONTH('2/20/2016') AS LastDay

Output :
sql server eomonth function

month_to_add optional parameter can be used to add or subtract a specified
number of months from the start_date, and then return the last day of the month from the resulting date.

The following example adds 2 months to the start_date and returns the last day of the month from the resulting date
SELECT EOMONTH('3/20/2016', 2) AS LastDay

Output :
ms sql server eomonth

The following example subtracts 1 month from the start_date and returns the last day of the month from the resulting date
SELECT EOMONTH('3/20/2016', -1) AS LastDay

Output :
sql server 2012 eomonth

Using EOMONTH function with table data. We will use the following Employees table for this example.
sql server 2012 eomonth example

SQL Script to create Employees table

```
Create table Employees
(
   Id int primary key identity,
   Name nvarchar(10),
   DateOfBirth date
)
Go

Insert into Employees values ('Mark', '01/11/1980')
Insert into Employees values ('John', '12/12/1981')
Insert into Employees values ('Amy', '11/21/1979')
Insert into Employees values ('Ben', '05/14/1978')
Insert into Employees values ('Sara', '03/17/1970')
Insert into Employees values ('David', '04/05/1978')
Go
```

The following example returns the last day of the month from the DateOfBirth of every employee.

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

sql server eomonth example

If you want just the last day instead of the full date, you can use DATEPART function

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

FROM Employees

ms sql server 2012 eomonth

--126-- DATEFROMPARTS FUNCTION

In this video we will discuss DATEFROMPARTS function in SQL Server

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

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

Output :
datefromparts function in sql server

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

SELECT DATEFROMPARTS (2015, 15, 25) AS [Date]

Output : Cannot construct data type date, some of the arguments have values which are not valid.

Example : NULL specified for month parameter, so the function returns NULL.

SELECT DATEFROMPARTS (2015, NULL, 25) AS [Date]

Output :
datefromparts in sql server

Other new date and time functions introduced in SQL Server 2012
EOMONTH (Discussed in Part 125 of SQL Server tutorial)
DATETIMEFROMPARTS : Returns DateTime
Syntax : DATETIMEFROMPARTS (year, month, day, hour, minute, seconds, milliseconds)
SMALLDATETIMEFROMPARTS : Returns SmallDateTime

Syntax : SMALLDATETIMEFROMPARTS (year, month, day, hour, minute)
We will discuss the following functions in a later video
TIMEFROMPARTS
DATETIME2FROMPARTS
DATETIMEOFFSETFROMPARTS
In our next video we will discuss the difference between DateTime and SmallDateTime.

--127- DIFFERENCE BETWEEN DATETIME AND SAMLLDATETIME IN SQL SERVER

In this video we will discuss the difference between DateTime and SmallDateTime in SQL Server

The following table summarizes the differences

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

The range for SmallDateTime is January 1, 1900, through June 6, 2079.
A value outside of this range, is not allowed.

The following 2 queries have values outside of the range of SmallDateTime data type.
Insert into Employees ([SmallDateTime]) values ('01/01/1899')
Insert into Employees ([SmallDateTime]) values ('07/06/2079')

When executed, the above queries fail with the following error
The conversion of a varchar data type to a smalldatetime data type resulted in an out-of-range value

The range for DateTime is January 1, 1753, through December 31, 9999. A value outside of this range, is not allowed.

The following query has a value outside of the range of DateTime data type.
Insert into Employees ([DateTime]) values ('01/01/1752')

When executed, the above query fails with the following error
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.

--128-- DATETIME2FROMPARTS FUNCTION IN SQL SERVER 2012

In this video we will discuss DateTime2FromParts function in SQL Server 2012.

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]

Output :
datetime2fromparts function in sql server 2012

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]

Output : 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]

Output :
datetimefromparts sql 2012

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

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

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

TIMEFROMPARTS : Returns time value

Syntax : TIMEFROMPARTS (hour, minute, seconds, fractions, precision)

Next video : We will discuss the difference between DateTime and DateTime2 in SQL Server

--129-- DIFFERENCE BETWEEN DATETIME AND DATETIME2 IN SQL SERVER

In this video we will discuss the difference between DateTime and DateTime2 in SQL Server

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.

DateTime2 Syntax : DATETIME2 [(fractional seconds precision)]

With DateTime2
Optional fractional seconds precision can be specified
The precision scale is from 0 to 7 digits
The default precision is 7 digits
For precision 1 and 2, storage size is 6 bytes
For precision 3 and 4, storage size is 7 bytes
For precision 5, 6 and 7, storage size is 8 bytes
The following script creates a table variable with 7 DATETIME2 columns with
different precision start from 1 through 7

```
DECLARE @TempTable TABLE
(
   DateTime2Precision1 DATETIME2(1),
   DateTime2Precision2 DATETIME2(2),
   DateTime2Precision3 DATETIME2(3),
   DateTime2Precision4 DATETIME2(4),
   DateTime2Precision5 DATETIME2(5),
   DateTime2Precision6 DATETIME2(6),
   DateTime2Precision7 DATETIME2(7)
)

Insert DateTime value into each column
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'
)

The following query retrieves the prcision, the datetime value, and the storage size.
SELECT 'Precision - 1' AS [Precision],
        DateTime2Precision1 AS DateValue,
        DATALENGTH(DateTime2Precision1) AS StorageSize
FROM @TempTable

UNION ALL

SELECT 'Precision - 2',
        DateTime2Precision2,
        DATALENGTH(DateTime2Precision2) AS StorageSize
FROM @TempTable

UNION ALL

SELECT 'Precision - 3',
        DateTime2Precision3,
        DATALENGTH(DateTime2Precision3)
FROM @TempTable

UNION ALL

SELECT 'Precision - 4',
        DateTime2Precision4,
        DATALENGTH(DateTime2Precision4)
FROM @TempTable

UNION ALL

SELECT 'Precision - 5',
        DateTime2Precision5,
        DATALENGTH(DateTime2Precision5)
FROM @TempTable

UNION ALL

SELECT 'Precision - 6',
        DateTime2Precision6,
        DATALENGTH(DateTime2Precision6)
FROM @TempTable

UNION ALL
SELECT 'Precision - 7',
        DateTime2Precision7,
```

```
        DATALENGTH(DateTime2Precision7) AS StorageSize
FROM @TempTable
```

Notice as the precision increases the storage size also increases
datetime2 precision scale

--130-- OFFSET FETCH NEXT IN SQL SERVER 2012

In this video we will discuss OFFSET FETCH Clause in SQL Server 2012

One of the common tasks for a SQL developer is to come up with a stored procedure that can return a page of results from the result set.
With SQL Server 2012 OFFSET FETCH Clause it is very easy to implement paging.

Lets understand this with an example. We will use the following tblProducts table for the examples in this video. The table has got 100 rows. In the image I have shown just 10 rows.

Offset fetch next in SQL Server 2012

SQL Script to create tblProducts table

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

SQL Script to populate tblProducts table with 100 rows

```
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 FETCH Clause
Introduced in SQL Server 2012
Returns a page of results from the result set
ORDER BY clause is required
OFFSET FETCH Syntax :

```
SELECT * FROM Table_Name
ORDER BY Column_List
OFFSET Rows_To_Skip ROWS
FETCH NEXT Rows_To_Fetch ROWS ONLY
```

The following SQL query
1. Sorts the table data by Id column
2. Skips the first 10 rows and
3. Fetches the next 10 rows

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

Result :
sql server 2012 offset fetch next

From the front-end application, we would typically send the PAGE NUMBER and the PAGE SIZE to get a page of rows. The following stored procedure accepts PAGE NUMBER and the PAGE SIZE as parameters and returns the correct set of rows.

```
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
```

With PageNumber = 3 and PageSize = 10, the stored procedure returns the correct set of rows

```
EXECUTE spGetRowsByPageNumberAndSize 3, 10
```

sql server 2012 paging stored procedure

--131-- IDENTIFYING OBJECT DEPENDENCIES IN SQL SERVER

In this video we will discuss how to identify object dependencies in SQL Server using SQL Server Management Studio.

The following SQL Script creates 2 tables, 2 stored procedures and a view

```
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
```

How to find dependencies using SQL Server Management Studio
Use View Dependencies option in SQL Server Management studio
to find the object dependencies

For example : To find the dependencies on the Employees table,
right click on it and select View Dependencies from the context menu

Identifying object dependencies in SQL Server

In the Object Dependencies window, depending on the radio button you select,
you can find the objects that depend on Employees table and the objects
on which Employees table depends on.

sql server object 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.

For example, there are 2 stored procedures (sp_GetEmployees and sp_GetEmployeesandDepartments)
that depend on the Employees table. If we are not aware of these dependencies
and if we delete the Employees table, both stored procedures will fail with the following error.

Msg 208, Level 16, State 1, Procedure sp_GetEmployees, Line 4
Invalid object name 'Employees'.

There are other ways for finding object dependencies in SQL Server
which we will discuss in our upcoming videos.

--132-- SYS.DM_SQL_REFERENCING_ENTITIES IN SQL SERVER

In this video we will discuss

How to find object dependencies using the following dynamic management functions
sys.dm_sql_referencing_entities
sys.dm_sql_referenced_entities
Difference between
Referencing entity and Referenced entity
Schema-bound dependency and Non-schema-bound dependency

This is continuation to Part 131, in which we discussed how to find object dependencies
using SQL Server Management Studio. Please watch Part 131 from SQL Server tutorial before
proceeding.

The following example 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.

To get the REFERENCING ENTITIES use SYS.DM_SQL_REFERENCING_ENTITIES dynamic
management function

To get the REFERENCED ENTITIES use SYS.DM_SQL_REFERENCED_ENTITIES dynamic
management function

Now, let us say we have a stored procedure and we want to find the all objects that this stored procedure depends on.
This can be very achieved using another dynamic management function, sys.dm_sql_referenced_entities.

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.
Without the schema name you may not get any results.

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 doesnt prevent the referenced object from
being dropped or modified.

--133-- SP_DEPENDS IN SQL SERVER

In this video we will discuss sp_depends system stored procedure.

There are several ways to find object dependencies in SQL Server
1. View Dependencies feature in SQL Server Management Studio - Part 131
2. SQL Server dynamic management functions - Part 132
    sys.dm_sql_referencing_entities
    sys.dm_sql_referenced_entities
3. sp_depends system stored procedure - This video

sp_depends
A system stored procedure that returns object dependencies
For example,
If you specify a table name as the argument, then the views and
procedures that depend on the specified table are displayed
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'

The following SQL Script creates a table and a stored procedure
Create table Employees
(
   Id int primary key identity,

```
    Name nvarchar(50),
    Gender nvarchar(10)
)
Go

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

Returns the stored procedure that depends on table Employees
sp_depends 'Employees'

Ouptut :
sp_depends in sql server

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

Output :
sql server sp depends stored procedure

Sometime sp_depends does not report dependencies correctly. For example, at
the moment we have Employees table and a stored procedure sp_GetEmployees.

Now drop the table Employees
Drop table Employees

and then recreate the table again

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

Now execute the following, to find the objects that depend on Employees table
sp_depends 'Employees'

We know that stored procedure sp_GetEmployees still depends on Employees table.
But sp_depends does not report this dependency, as the Employees table is dropped
and recreated.
Object does not reference any object, and no objects reference it.

sp_depends is on the deprecation path. This might be removed from the future versions of SQL server.

--134-- SEQUENCE OBJECT IN SQL SERVER 2012

In this video we will discuss sequence object in SQL Server.

Sequence object
Introduced in SQL Server 2012
Generates sequence of numeric values in an ascending or descending order
Syntax :

```
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.

Creating an Incrementing Sequence : The following code create a Sequence object that starts with 1 and increments by 1

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 1
INCREMENT BY 1
```

Generating the Next Sequence Value : Now we have a sequence object created. To generate the sequence value use NEXT VALUE FOR clause

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

Output : 1

Every time you execute the above query the sequence value will be incremented by 1.

I executed the above query 5 times, so the current sequence value is 5.

Retrieving the current sequence value :
If you want to see what the current Sequence value before generating the next, use sys.sequences

```
SELECT * FROM sys.sequences WHERE name = 'SequenceObject'
```

Alter the Sequence object to reset the sequence value :

```
ALTER SEQUENCE [SequenceObject] RESTART WITH 1
```

Select the next sequence value to make sure the value starts from 1

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

Using sequence value in an INSERT query :

```
CREATE TABLE Employees
(
   Id INT PRIMARY KEY,
   Name NVARCHAR(50),
   Gender NVARCHAR(10)
)

-- Generate and insert Sequence values
INSERT INTO Employees VALUES
(NEXT VALUE for [dbo].[SequenceObject], 'Ben', 'Male')
INSERT INTO Employees VALUES
(NEXT VALUE for [dbo].[SequenceObject], 'Sara', 'Female')

-- Select the data from the table

SELECT * FROM Employees
```

sequence object in sql server 2012

Creating the decrementing Sequence : The following code create a Sequence object that starts with 100 and decrements by 1

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 100
INCREMENT BY -1
```

Specifying MIN and MAX values for the sequence : Use the MINVALUE and MAXVALUE arguments to specify the MIN and MAX values respectively.

Step 1 : Create the Sequence object

```
CREATE SEQUENCE [dbo].[SequenceObject]
  START WITH 100
  INCREMENT BY 10
```

```
  MINVALUE 100
  MAXVALUE 150
```

Step 2 : Retrieve the next sequence value. The sequence value starts at 100. Every time we call NEXT VALUE, the value will be incremented by 10.

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

If you call NEXT VALUE, when the value reaches 150 (MAXVALUE), you will get the following error
The sequence object 'SequenceObject' has reached its minimum or maximum value. Restart the sequence object to allow new values to be generated.

Recycling Sequence values : When the sequence object has reached its 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
```

At this point, whe the sequence object has reached its maximum value, and if you ask for the NEXT VALUE,
sequence object starts from the minimum value again which in this case is 100.

To improve performance, the Sequence object values can be cached using the CACHE option.
When the values are cached they are read from the memory instead of from the disk, which improves the performance.
When the cache option is specified you can also specify the size of th cache , that is the number of values to cache.

The following example, creates the sequence object with 10 values cached.
When the 11th value is requested, the next 10 values will be cached again.

```
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

create sequence in sql server

Next video : Difference between SEQUENCE and IDENTITY in SQL Server

--135-- DIFFERENCE BETWEEN SEQUENCE AND IDENTITY IN SQL SERVER

In this video we will discuss the difference between SEQUENCE and IDENTITY in SQL Server

This is continuation to Part 134. Please watch Part 134 from SQL Server tutorial before proceeding.

Sequence object is similar to the Identity property, in the sense that it generates sequence of numeric values in an ascending order just like the identity property. However there are several differences between the 2 which we will discuss in this video.

Identity property is a table column property meaning it is tied to the table, where as the sequence is a user-defined database object and is not tied to any specific table meaning its value can be shared by multiple tables.

Example : Identity property tied to the Id column of the Employees table.

```
CREATE TABLE Employees
(
   Id INT PRIMARY KEY IDENTITY(1,1),
   Name NVARCHAR(50),
   Gender NVARCHAR(10)
)
```

Example : Sequence object not tied to any specific table

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 1
INCREMENT BY 1
```

This means the above sequence object can be used with any table.

Example : Sharing sequence object value with multiple tables.

Step 1 : Create Customers and Users tables

```
CREATE TABLE Customers
(
   Id INT PRIMARY KEY,
   Name NVARCHAR(50),
   Gender NVARCHAR(10)
)
GO
CREATE TABLE Users
(
   Id INT PRIMARY KEY,
```

```
    Name NVARCHAR(50),
    Gender NVARCHAR(10)
)
GO
```

Step 2 : Insert 2 rows into Customers table and 3 rows into Users table.
Notice the same sequence object is generating the ID values for both the tables.

```
INSERT INTO Customers VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Ben', 'Male')
INSERT INTO Customers VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Sara', 'Female')

INSERT INTO Users VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Tom', 'Male')
INSERT INTO Users VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Pam', 'Female')
INSERT INTO Users VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'David', 'Male')
GO
```

Step 3 : Query the tables

```
SELECT * FROM Customers
SELECT * FROM Users
GO
```

Output : Notice the same sequence object has generated the
values for ID columns in both the tables

Difference between sequence and identity in SQL Server

To generate the next identity value, a row has to be inserted into the table, where as 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.

Example : Generating Identity value by inserting a row into the table

```
INSERT INTO Employees VALUES ('Todd', 'Male')
```

Example : Generating the next sequence value using NEXT VALUE FOR clause.

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

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 its data type.

Example : Specifying maximum value for the sequence object using the MAXVALUE option

```
CREATE SEQUENCE [dbo].[SequenceObject]
START WITH 1
INCREMENT BY 1
MAXVALUE 5
```

CYCLE option of the Sequence object 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, where as with the Identity property we dont have any such option to automatically restart the identity values.

Example : Specifying the CYCLE option of the Sequence object, so the sequence will restart automatically when the max value is exceeded

```
CREATE SEQUENCE [dbo].[SequenceObject]
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 5
CYCLE
```

--136-- GUID IN SQL SERVER

What is in SQL Server
The GUID data type 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.

To declare a GUID variable, we use the keyword UNIQUEIDENTIFIER

```
Declare @ID UNIQUEIDENTIFIER
SELECT @ID = NEWID()
SELECT @ID as MYGUID
```

How to create a GUID in sql server
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. Every time you execute SELECT NEWID() query, you get a GUID that is unique.

Example GUID : 0BB83607-00D7-4B2C-8695-32AD3812B6F4

When to use GUID data type : Let us understand when to use a GUID in SQL Server with an example.

1. Let us say our company does business in 2 countries - USA and India.

2. USA customers are stored in a table called USACustomers in a database called USADB.

```
Create Database USADB
Go

Use USADB
Go

Create Table USACustomers
(
    ID int primary key identity,
    Name nvarchar(50)
)
Go

Insert Into USACustomers Values ('Tom')
Insert Into USACustomers Values ('Mike')

Select * From USADB.dbo.USACustomers
```

The above query produces the following data
sql server create guid

3. India customers are stored in a table called IndiaCustomers in a
database called IndiaDB.

```
Create Database USADB
Go

Use USADB
Go

Create Table USACustomers
(
    ID int primary key identity,
    Name nvarchar(50)
)
Go
```

Insert Into USACustomers Values ('Tom')
Insert Into USACustomers Values ('Mike')

Select * From USADB.dbo.USACustomers

The above query produces the following data
guid data type sql server

In both the tables, the ID column data type is integer. It is also the primary key column which ensures the ID
column across every row is unique in that table. We also have turned on the identity property,

4. Now, we want to load the customers from both countries (India & USA) in to a single existing table Customers.

First lets create the table. Use the following SQL script to create the table. ID column is the primary key of the table.

```
Create Table Customers
(
    ID int primary key,
    Name nvarchar(50)
)

Go
```

Now execute the following script which selects the data from IndiaCustomers and USACustomers tables and inserts into Customers table

```
Insert Into Customers
Select * from IndiaDB.dbo.IndiaCustomers
Union All
Select * from USADB.dbo.USACustomers
```

We get the following error. This is because in both the tables, Identity column data type is integer. Integer is great for identity as
long as you only want to maintain the uniqueness across just that one table. However, between IndiaCustomers and USACustomers tables,
the ID coulumn values are not unique. So when we load the data into Customers table, we get "Violation of PRIMARY KEY constraint" error.

```
Msg 2627, Level 14, State 1, Line 1
Violation of PRIMARY KEY constraint. Cannot insert duplicate key in object 'dbo.Customers'. The duplicate key value is (1).
The statement has been terminated.
```

A GUID on the other hand is unique across tables, databases, and servers. A GUID is guaranteed to be globally unique.
Let us see if we can solve the above problem using a GUID.

Create USACustomers1 table and populate it with data. Notice ID column datatype is uniqueidentifier. To auto-generate the GUID values
we are using a Default constraint.

```
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')
```

Next, create IndiaCustomers1 table and populate it with data.

```
Use IndiaDB
Go

Create Table IndiaCustomers1
(
    ID uniqueidentifier primary key default NEWID(),
    Name nvarchar(50)
)
Go

Insert Into IndiaCustomers1 Values (Default, 'John')
Insert Into IndiaCustomers1 Values (Default, 'Ben')
```

Select data from both the tables (USACustomers1 & IndiaCustomers1). Notice the ID column values. They are unique across both the tables.

```
Select * From IndiaDB.dbo.IndiaCustomers1
UNION ALL
Select * From USADB.dbo.USACustomers1
```

uniqueidentifier in sql server

Now, we want to load the customers from USACustomers1 and IndiaCustomers1 tables in to a single existing table called Customers1.
Let us first create Customers1 table. The ID column in Customers1 table is uniqueidentifier.

```
Create Table Customers1
(
    ID uniqueidentifier primary key,
```

```
    Name nvarchar(50)
)
Go
```

Finally, execute the following insert script. Notice the script executes successfully without any errors and the data is loaded into
Customers1 table.

```
Insert Into Customers1
Select * from IndiaDB.dbo.IndiaCustomers1
Union All
Select * from USADB.dbo.USACustomers1
```

The main advantage of using a GUID is that it is unique across tables, databases and servers. It is extremely useful if youre consolidating
records from multiple SQL Servers into a single table.

The main disadvantage of using a GUID as a key is that it is 16 bytes in size. It is one of the largest datatypes in SQL Server.
An integer on the other hand is 4 bytes,

An Index built on a GUID is larger and slower than an index built on integer column. In addition a GUID is hard to read compared to int.

So in summary, use a GUID when you really need a globally unique identifier. In all other cases it is better to use an INT data type.

--137-- HOW TO CHECK GUID IS NULL OR EMPTY IN SQL SERVER

In this video we will discuss how to check if a GUID is null or empty

How to check if a GUID is NULL : Checking if a GUID is null is straight forward in SQL Server. Just use IS NULL keywords as shown below.

```
Declare @MyGuid Uniqueidentifier

If(@MyGuid IS NULL)
Begin
    Print 'Guid is null'
End
Else
Begin
    Print 'Guid is not null'
End
```

In the above example, since @MyGuid is just declared and not initialised, it prints the message "Guid is null"

Now lets say, if a GUID variable is NULL, then we want to initialise that GUID variable with a new GUID value.
If its not NULL, then we want to retain its value. One way to do this is by using an IF condition as shown below.

```
Declare @MyGuid UniqueIdentifier

If(@MyGuid IS NULL)
Begin
    Set @MyGuid = NEWID()
End
Select @MyGuid
```

We can achieve exactly the same thing by using ISNULL() function. The advantage of using ISNULL() function is that,
it reduces the amount of code we have to write.

```
Declare @MyGuid UniqueIdentifier
Select ISNULL(@MyGuid, NewID())
```

How to check if a GUID is EMPTY : Before understanding how to check if a GUID is empty, lets understand what is an empty GUID.
An empty GUID is a GUID with all ZEROS as shown below.
00000000-0000-0000-0000-000000000000

How to create this empty GUID. Do we have to type all the ZEROs and Hyphens. The answe is NO. We do not have to type them manually.
Instead use one of the following SELECT querys to create an empty GUID. I prefer to use the second SELECT statement as it has only one CAST

```
SELECT CAST(CAST(0 AS BINARY) AS UNIQUEIDENTIFIER)
OR
SELECT CAST(0x0 AS UNIQUEIDENTIFIER)
```

To check if a GUID is an empty GUID, you have 2 options
Option 1: You can compare it to an Empty GUID value as shown below

```
Declare @MyGuid UniqueIdentifier
Set @MyGuid = '00000000-0000-0000-0000-000000000000'

If(@MyGuid = '00000000-0000-0000-0000-000000000000')
Begin
    Print 'Guid is Empty'
End
Else
Begin
```

```
    Print 'Guid is not Empty'
End
```

Option 2: You can also compare it to a return value of the CAST method

```
Declare @MyGuid UniqueIdentifier
Set @MyGuid = '00000000-0000-0000-0000-000000000000'

If(@MyGuid = Cast(0x0 as Uniqueidentifier))
Begin
    Print 'Guid is Empty'
End
Else
Begin
    Print 'Guid is not Empty'
End
```

--138-- DYNAMIC SQL IN SQL SERVER

What is Dynamic SQL
Dynamic SQL is a SQL built from strings at runtime.

Simple example of using Dynamic SQL : Lets say we want to
implement "Employee Search" web page as shown below.
dynamic sql sql server

Depending on the serach fields the end user provides, we want to search the following Employees table.
dynamic sql sql server tutorial

Here is the SQL Script to create Employees table and populate it with data

```
Create table Employees
(
    ID int primary key identity,
    FirstName nvarchar(50),
    LastName nvarchar(50),
    Gender nvarchar(50),
    Salary int
)
Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)
Insert into Employees values ('Steve', 'Pound', 'Male', 45000)
Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)
```

```
Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)
Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)
Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)
Insert into Employees values ('John', 'Stanmore', 'Male', 80000)
Go
```

One way to achieve this is by implementing a stored procedure as shown below that this page would call.

```
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
Go
```

The stored procedure in this case is not very complicated as we have only 4 search filters.
What if there are 20 or more such filters. This stored procedure can get complex.
To make things worse what if we want to specify conditions like AND, OR etc between these search filters. The stored procedure can get extremely large, complicated and difficult to maintain.
One way to reduce the complexity is by using dynamic SQL as show below.
Depending on for which search filters the user has provided the values on the "Search Page", we build the WHERE clause dynamically at runtime, which can reduce complexity.

However, you might hear arguments that dynamic sql is bad both in-terms of security and performance. This is true if the dynamic sql is not properly implemented. From a security standpoint, it may open doors for SQL injection attack and from a performance standpoint, the cached query plans may not be reused. If properly implemented, we will not have these problems with dynamic sql.
In our upcoming videos, we will discuss good and bad dynamic sql implementations.

For now lets implement a simple example that makes use of dynamic sql.
In the example below we are assuming the user has supplied values only for FirstName and LastName search fields.
To execute the dynamicl sql we are using system stored procedure sp_executesql.

sp_executesql takes two pre-defined parameters and any number of user-defined parameters.

@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 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)'

Execute sp_executesql @sql, @params, @FirstName='Ben',@LastName='Hoskins'
```

This is just the introduction to dynamic SQL. If a few things are unclear at the moment, dont worry.
In our upcoming videos we will discuss the following
Implementing a real world "Search Web Page" with and without dynamic SQL
Performance and Security implications of dynamic sql. Along the way we will also discuss good and bad dynamic sql implementations.
Different options available for executing dynamic sql and their implications
Using dynamic sql in stored procedures and its implications
Once we discuss all the above, you will understand
1. The flexibility dynamic sql provides
2. Advantages and disadvantages of dynamic sql
3. When and when not to use dynamic sql

--139-- IMPLEMENT SEARCH WEB PAGE USING ASP.NET AND STORED PROCEDURE

In this video we will discuss implementing a search web page using ASP.NET and Stored Procedure. This is continuation to Part 138. Please watch Part 138 from SQL Server Tutorial before proceeding.

The search page looks as shown below.
how to search data using stored procedure in asp.net c#

Step 1 : Modify the "spSearchEmployees" stored procedure to include NULL as the default value for the parameters. The advantage of specifying default value for the parameters is that the ASP.NET page need not pass those parameters when calling the stored procedures if the user did not specify any values for the corresponding search fields on the Search Page.

```
Alter Procedure spSearchEmployees
@FirstName nvarchar(100) = NULL,
@LastName nvarchar(100) = NULL,
@Gender nvarchar(50) = NULL,
@Salary int = NULL
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
Go
```

Step 2 : Create a new empty ASP.NET Web Forms application. Name it "DynamicSQLDemo".

Step 3 : Add the connection string to your database in web.config

```
<add name="connectionStr"
     connectionString="server=.;database=EmployeeDB;integrated security=true"/>
```

Step 4 : Add a WebForm to the project. Name it "SearchPageWithoutDynamicSQL.aspx"

Step 5 : Copy and paste the following HTML on the ASPX page. Notice we are using Bootstrap to style the page. If you are new to Bootstrap, please check out our Bootstrap tutorial for beginners playlist.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Employee Search</title>
   <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
      type="text/css" />
</head>
<body style="padding-top: 10px">
   <div class="col-xs-8 col-xs-offset-2">
      <form id="form1" runat="server" class="form-horizontal">
         <div class="panel panel-primary">
            <div class="panel-heading">
               <h3>Employee Search Form</h3>
            </div>
            <div class="panel-body">
               <div class="form-group">
                  <label for="inputFirstname" class="control-label col-xs-2">
                     Firstname
                  </label>
                  <div class="col-xs-10">
                     <input type="text" runat="server" class="form-control"
                        id="inputFirstname" placeholder="Firstname" />
                  </div>
               </div>

               <div class="form-group">
                  <label for="inputLastname" class="control-label col-xs-2">
```

```
                Lastname
            </label>
            <div class="col-xs-10">
                <input type="text" runat="server" class="form-control"
                    id="inputLastname" placeholder="Lastname" />
            </div>
        </div>

        <div class="form-group">
            <label for="inputGender" class="control-label col-xs-2">
                Gender
            </label>
            <div class="col-xs-10">
                <input type="text" runat="server" class="form-control"
                    id="inputGender" placeholder="Gender" />
            </div>
        </div>

        <div class="form-group">
            <label for="inputSalary" class="control-label col-xs-2">
                Salary
            </label>
            <div class="col-xs-10">
                <input type="number" runat="server" class="form-control"
                    id="inputSalary" placeholder="Salary" />
            </div>
        </div>
        <div class="form-group">
            <div class="col-xs-10 col-xs-offset-2">
                <asp:Button ID="btnSearch" runat="server" Text="Search"
                    CssClass="btn btn-primary" OnClick="btnSearch_Click" />
            </div>
        </div>
    </div>
</div>

<div class="panel panel-primary">
    <div class="panel-heading">
        <h3>Search Results</h3>
    </div>
    <div class="panel-body">
        <div class="col-xs-10">
            <asp:GridView CssClass="table table-bordered"
                ID="gvSearchResults" runat="server">
            </asp:GridView>
        </div>
    </div>
</div>
</form>
```

```
    </div>
</body>
</html>
```

Step 6 : Copy and paste the following code in the code-behind page. Notice we are using the stored procedure "spSearchEmployees".
We are not using any dynamic SQL in this example. In our next video, we will discuss implementing the same "Search Page" using
dynamic sql and understand the difference between using dynamic sql and stored procedure.

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

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

        protected void btnSearch_Click(object sender, EventArgs e)
        {
            string connectionStr = ConfigurationManager
                .ConnectionStrings["connectionStr"].ConnectionString;
            using(SqlConnection con = new SqlConnection(connectionStr))
            {
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = con;
                cmd.CommandText = "spSearchEmployees";
                cmd.CommandType = CommandType.StoredProcedure;

                if(inputFirstname.Value.Trim() != "")
                {
                    SqlParameter param = new SqlParameter
                        ("@FirstName", inputFirstname.Value);
                    cmd.Parameters.Add(param);
                }

                if (inputLastname.Value.Trim() != "")
                {
                    SqlParameter param = new SqlParameter
                        ("@LastName", inputLastname.Value);
                    cmd.Parameters.Add(param);
                }

                if (inputGender.Value.Trim() != "")
                {
```

```
            SqlParameter param = new SqlParameter
                ("@Gender", inputGender.Value);
            cmd.Parameters.Add(param);
        }

        if (inputSalary.Value.Trim() != "")
        {
            SqlParameter param = new SqlParameter
                ("@Salary", inputSalary.Value);
            cmd.Parameters.Add(param);
        }

        con.Open();
        SqlDataReader rdr = cmd.ExecuteReader();
        gvSearchResults.DataSource = rdr;
        gvSearchResults.DataBind();
    }
  }
 }
}
```

--140-- IMPLEMENT SEARCH WEB PAGE USING ASP.NET AND DYNAMIC SQL

In this video we will discuss implementing a search web page using ASP.NET and Dynamic SQL.
This is continuation to Part 139. Please watch Part 139 from SQL Server Tutorial before proceeding.

Step 1 : Add a WebForm to the web project. Name it "SearchPageWithDynamicSQL.aspx"

Step 2 : Copy and paste the following HTML on the ASPX page. Notice we are using Bootstrap to style the page.
If you are new to Bootstrap, please check out our Bootstrap tutorial for beginners playlist.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Employee Search</title>
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    type="text/css" />
</head>
<body style="padding-top: 10px">
  <div class="col-xs-8 col-xs-offset-2">
    <form id="form1" runat="server" class="form-horizontal">
      <div class="panel panel-primary">
        <div class="panel-heading">
          <h3>Employee Search Form</h3>
        </div>
        <div class="panel-body">
          <div class="form-group">
            <label for="inputFirstname" class="control-label col-xs-2">
```

```
            Firstname
          </label>
          <div class="col-xs-10">
            <input type="text" runat="server" class="form-control"
              id="inputFirstname" placeholder="Firstname" />
          </div>
        </div>

        <div class="form-group">
          <label for="inputLastname" class="control-label col-xs-2">
            Lastname
          </label>
          <div class="col-xs-10">
            <input type="text" runat="server" class="form-control"
              id="inputLastname" placeholder="Lastname" />
          </div>
        </div>

        <div class="form-group">
          <label for="inputGender" class="control-label col-xs-2">
            Gender
          </label>
          <div class="col-xs-10">
            <input type="text" runat="server" class="form-control"
              id="inputGender" placeholder="Gender" />
          </div>
        </div>

        <div class="form-group">
          <label for="inputSalary" class="control-label col-xs-2">
            Salary
          </label>
          <div class="col-xs-10">
            <input type="number" runat="server" class="form-control"
              id="inputSalary" placeholder="Salary" />
          </div>
        </div>
        <div class="form-group">
          <div class="col-xs-10 col-xs-offset-2">
            <asp:Button ID="btnSearch" runat="server" Text="Search"
              CssClass="btn btn-primary" OnClick="btnSearch_Click" />
          </div>
        </div>
      </div>
    </div>

    <div class="panel panel-primary">
      <div class="panel-heading">
        <h3>Search Results</h3>
```

```
            </div>
            <div class="panel-body">
               <div class="col-xs-10">
                  <asp:GridView CssClass="table table-bordered"
                     ID="gvSearchResults" runat="server">
                  </asp:GridView>
               </div>
            </div>
         </div>
      </form>
   </div>
</body>
</html>
```

Step 3 : Copy and paste the following code in the code-behind page. Notice we are using dynamic sql instead of the stored procedure
"spSearchEmployees".

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

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

      protected void btnSearch_Click(object sender, EventArgs e)
      {
         string strConnection = ConfigurationManager
            .ConnectionStrings["connectionStr"].ConnectionString;

         using (SqlConnection con = new SqlConnection(strConnection))
         {
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = con;

            StringBuilder sbCommand = new
               StringBuilder("Select * from Employees where 1 = 1");

            if (inputFirstname.Value.Trim() != "")
            {
               sbCommand.Append(" AND FirstName=@FirstName");
               SqlParameter param = new
                  SqlParameter("@FirstName", inputFirstname.Value);
```

```
            cmd.Parameters.Add(param);
        }

        if (inputLastname.Value.Trim() != "")
        {
            sbCommand.Append(" AND LastName=@LastName");
            SqlParameter param = new
                SqlParameter("@LastName", inputLastname.Value);
            cmd.Parameters.Add(param);
        }

        if (inputGender.Value.Trim() != "")
        {
            sbCommand.Append(" AND Gender=@Gender");
            SqlParameter param = new
                SqlParameter("@Gender", inputGender.Value);
            cmd.Parameters.Add(param);
        }

        if (inputSalary.Value.Trim() != "")
        {
            sbCommand.Append(" AND Salary=@Salary");
            SqlParameter param = new
                SqlParameter("@Salary", inputSalary.Value);
            cmd.Parameters.Add(param);
        }

        cmd.CommandText = sbCommand.ToString();
        cmd.CommandType = CommandType.Text;

        con.Open();
        SqlDataReader rdr = cmd.ExecuteReader();
        gvSearchResults.DataSource = rdr;
        gvSearchResults.DataBind();
    }
  }
}
```

At this point, run the application and SQL profiler. To run SQL profiler
1. Open SQL Server Management Studio
2. Click on "Tools" and select "SQL Server Profiler"
3. Click the "Connect" button to connect to local SQl Server instance
4. Leave the "Defaults" on "Trace Properties" window and click on "Run" button
5. We now have the SQL Profiler running and in action

On the "Search Page" set "Gender" filter to Male and click the "Search" button.
Notice we get all the Male employees as expected.
Also in the SQL Server profiler you can see the Dynamic SQL statement is

executed using system stored procedure sp_executesql.
sql server profiler exec sp_executesql

In our next video, we will discuss the differences between using Dynamic SQL and Stored Procedures

--141-- PREVENT SQL INJECTIO WITH DYNAMIC SQL

In this video we will discuss, how to prevent SQL injection when using dynamic SQL.
This is continuation to Part 140. Please watch Part 140 from SQL Server Tutorial before proceeding.

In Part 140, we have implemented "Search Page" using dynamic SQL. Since we have used parameters to
build our dynamic SQL statements, it is not prone to SQL Injection attack.
This is an example of good dynamic SQL implementation.

I have seen lot of software developers, not just the beginners but even experienced developers, buidling their dynamic sql queries by concatenating strings instead of using parameters without realizing that they are opening the doors for SQL Injection.

Here is an example of bad dynamic SQL that is prone to SQL Injection

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

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

      protected void btnSearch_Click(object sender, EventArgs e)
      {
         string strConnection = ConfigurationManager
            .ConnectionStrings["connectionStr"].ConnectionString;

         using (SqlConnection con = new SqlConnection(strConnection))
         {
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = con;

            StringBuilder sbCommand = new
               StringBuilder("Select * from Employees where 1 = 1");

            if (inputFirstname.Value.Trim() != "")
            {
```

```
            sbCommand.Append(" AND FirstName = '" +
                inputFirstname.Value + "'");
        }


        if (inputLastname.Value.Trim() != "")
        {
            sbCommand.Append(" AND LastName = '" +
                inputLastname.Value + "'");
        }

        if (inputGender.Value.Trim() != "")
        {
            sbCommand.Append(" AND Gender = '" +
                inputGender.Value + "'");
        }

        if (inputSalary.Value.Trim() != "")
        {
            sbCommand.Append(" AND Salary = " + inputSalary.Value);
        }

        cmd.CommandText = sbCommand.ToString();
        cmd.CommandType = CommandType.Text;

        con.Open();
        SqlDataReader rdr = cmd.ExecuteReader();
        gvSearchResults.DataSource = rdr;
        gvSearchResults.DataBind();
      }
    }
  }
}
```

Since we are concatenating the user input values to build the dynamic sql statement, the end user can very easily inject sql.
Imagine, what happens for example, if the user enters the following in the "Firstname" textbox.

```
Drop database SalesDB --
```

With the above SQL injected into the "Firstname" textbox, if you click the "Search" button, the following is the query which
is sent to SQL server. This will drop SalesDB database.

```
Select * from Employees where 1 = 1 AND FirstName = '' Drop database SalesDB --'
```

On the other hand, if you use parameters to build your dynamic SQL statements, SQL Injection is not an issue.
The following input in the "Firstname" textbox, would not drop the SalesDB database.

Drop database SalesDB --

The text the user has provided in the "Firstname" textbox is treated as the value for @Firstname parameter.
The following is the query that is generated and executed.

exec sp_executesql N'Select * from Employees where 1 = 1 AND
FirstName=@FirstName',N'@FirstName nvarchar(26)',@FirstName=N'"
Drop database SalesDB --'

We dont have this problem of sql injection if we are using stored procedures.
"SearchPageWithoutDynamicSQL.aspx"
is using the stored procedure "spSearchEmployees" instead of dynamic SQL. The same input in the "Firstname" texbox on
this page, would generate the following. Notice, whatever text we typed in the "Firstname" textbox is treated as the
value for @FirstName parameter.

exec spSearchEmployees @FirstName=N'" Drop database SalesDB --'

An important point to keep in mind here is that if you have dynamic SQL in your stored procedure, and you are
concatenating strings in that stored procedure to build your dynamic sql statements instead of using parameters, it is
still prone to SQL injection. If this is not clear at the moment dont worry, we will discuss an example of this
in out next video.

So in summary, while dynamic sql provides great flexibility when implementing complicated logic with lot of
permutations and combinations, if not properly implemented it may open doors for sql injection.
Always use parameters to build dynamic sql statements, instead of concatenating user input values.

Another benefit of using parameters to build dynamic sql statements is that it allows cached query plans to be reused,
which greatly increases the performance. We will discuss an example of this in our upcoming videos.

--142-- SQL SERVER QUERY PLAN CACHE

In this video we will discuss
1. What happens when a query is issued to SQL Server
2. How to check what is in SQL Server plan cache
3. Things to consider to promote query plan reusability

What happens when a query is issued to SQL Server

In SQl Server, every query requires a query plan before it is executed. When you run a query the first time,
the query gets compiled and a query plan is generated. This query plan is then saved in sql server query plan cache.

Next time when we run the same query, the cached query plan is reused. This means sql server does not have to
create the plan again for that same query. So reusing a query plan can increase the performance.

How long the query plan stays in the plan cache depends on how often the plan is reused besides other factors.
The more often the plan is reused the longer it stays in the plan cache.

How to check what is in SQL Server plan cache
To see what is in SQL Server plan cache we will make use of the following 3 dynamic management views and functions provided by sql server
1. sys.dm_exec_cached_plans
2. sys.dm_exec_sql_text
3. sys.dm_exec_query_plan

Please note we discussed CROSS APPLY in detail in Part 91 of SQL Server tutorial.

Use the following query to see what is in the plan cache

```
SELECT cp.usecounts, cp.cacheobjtype, cp.objtype, st.text, qp.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC
```

As you can see we have sorted the result set by usecounts column in descending order, so we can see the most
frequently reused query plans on the top. The output of the above query from my computer is shown below.
Sql server query plan cache

The following table explains what each column in the resultset contains

Column	Description
usecounts	Number of times the plan is reused
objtype	Specifies the type of object
text	Text of the SQL query
query_plan	Query execution plan in XML format

To remove all elements from the plan cache use the following command
DBCC FREEPROCCACHE

In older versions of SQL Server up to SQL Server 6.5 only stored procedure plans are cached.

The query plans for Adhoc sql statements or dynamic sql statements are not cached, so they get compiled every time.
With SQL Server 7, and later versions the query plans for Adhoc sql statements and dynamic sql statements are also cached.

Things to consider to promote query plan reusability

For example, when we execute the following query the first time. The query is compiled, a plan is created and put in the cache.
Select * From Employees Where FirstName = 'Mark'

When we execute the same query again, it looks up the plan cache, and if a plan is available, it reuses the existing plan instead
of creating the plan again which can improve the performance of the query. However, one important thing to keep in mind is that,
the cache lookup is by a hash value computed from the query text. If the query text changes even slightly, sql server will not be
able to reuse the existing plan.

For example, even if you include an extra space somewhere in the query or you change the case, the query text hash will not match,
and sql server will not be able find the plan in cache and ends up compiling the query again and creating a new plan.

Another example : If you want the same query to find an employee whose FirstName is Steve instead of Mark. You would issue the following query
Select * From Employees Where FirstName = 'Steve'

Even in this case, since the query text has changed the hash will not match, and sql server will not be able find the plan in
cache and ends up compiling the query again and creating a new plan.

This is why, it is very important to use parameterised queries for sql server to be able to reuse cached query plans.
With parameterised queries, sql server will not treat parameter values as part of the query text.
So when you change the parameters values, sql server can still reuse the cached query plan.

The following query uses parameters. So even if you change parameter values, the same query plan is reused.

```
Declare @FirstName nvarchar(50)
Set @FirstName = 'Steve'
Execute sp_executesql N'Select * from Employees where FirstName=@FN', N'@FN nvarchar(50)',
@FirstName
```

One important thing to keep in mind is that, when you have dynamic sql in a stored procedure, the query plan for the stored
procedure does not include the dynamic SQL. The block of dynamic SQL has a query plan of its own.

Summary: Never ever concatenate user input values with strings to build dynamic sql statements. Always use parameterised queries which not only promotes cached query plans reuse but also prevent sql injection attacks.

--143-- EXEC VS SP_EXECUTESQL IN SQL SERVER

In this video we will discuss
1. What happens when a query is issued to SQL Server
2. How to check what is in SQL Server plan cache
3. Things to consider to promote query plan reusability

What happens when a query is issued to SQL Server
In SQl Server, every query requires a query plan before it is executed. When you run a query the first time,
the query gets compiled and a query plan is generated. This query plan is then saved in sql server query plan cache.

Next time when we run the same query, the cached query plan is reused. This means sql server does not have to
create the plan again for that same query. So reusing a query plan can increase the performance.

How long the query plan stays in the plan cache depends on how often the plan is reused besides other factors.
The more often the plan is reused the longer it stays in the plan cache.

How to check what is in SQL Server plan cache
To see what is in SQL Server plan cache we will make use of the following 3 dynamic management views and functions provided by sql server
1. sys.dm_exec_cached_plans
2. sys.dm_exec_sql_text
3. sys.dm_exec_query_plan

Please note we discussed CROSS APPLY in detail in Part 91 of SQL Server tutorial.

Use the following query to see what is in the plan cache

```
SELECT cp.usecounts, cp.cacheobjtype, cp.objtype, st.text, qp.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC
```

As you can see we have sorted the result set by usecounts column in descending order,
so we can see the most frequently reused query plans on the top.

The output of the above query from my computer is shown below.
Sql server query plan cache

The following table explains what each column in the resultset contains

Column	Description
usecounts	Number of times the plan is reused
objtype	Specifies the type of object
text	Text of the SQL query
query_plan	Query execution plan in XML format

To remove all elements from the plan cache use the following command
DBCC FREEPROCCACHE

In older versions of SQL Server up to SQL Server 6.5 only stored procedure plans are cached.
The query plans for Adhoc sql statements or dynamic sql statements are not cached,
so they get compiled every time.
With SQL Server 7, and later versions the query plans for Adhoc sql statements and
dynamic sql statements are also cached.

Things to consider to promote query plan reusability

For example, when we execute the following query the first time.
The query is compiled, a plan is created and put in the cache.
Select * From Employees Where FirstName = 'Mark'

When we execute the same query again, it looks up the plan cache,
and if a plan is available, it reuses the existing plan instead of
creating the plan again which can improve the performance of the query.
However, one important thing to keep in mind is that,
the cache lookup is by a hash value computed from the query text.
If the query text changes even slightly,
sql server will not be able to reuse the existing plan.

For example, even if you include an extra space somewhere in the query or
you change the case, the query text hash will not match,
and sql server will not be able find the plan in cache and ends up
compiling the query again and creating a new plan.

Another example : If you want the same query to find an employee whose
FirstName is Steve instead of Mark. You would issue the following query
Select * From Employees Where FirstName = 'Steve'

Even in this case, since the query text has changed the hash will not match,
and sql server will not be able find the plan
in cache and ends up compiling the query again and creating a new plan.

This is why, it is very important to use parameterised queries for
sql server to be able to reuse cached query plans.
With parameterised queries, sql server will not treat parameter values as part of the query text.

So when you change the parameters values, sql server can still reuse the cached query plan.

The following query uses parameters.
So even if you change parameter values, the same query plan is reused.

```
Declare @FirstName nvarchar(50)
Set @FirstName = 'Steve'
Execute sp_executesql N'Select * from Employees where FirstName=@FN', N'@FN nvarchar(50)',
@FirstName
```

One important thing to keep in mind is that, when you have dynamic sql in a stored procedure, the query plan for the
stored procedure does not include the dynamic SQL. The block of dynamic SQL has a query plan of its own.

Summary: Never ever concatenate user input values with strings to build dynamic sql statements. Always use parameterised queries which not only promotes cached query plans reuse but also prevent sql injection attacks.

--144-- DYNAMIC SQL TABLE NAME VARIABLE

In this video we will discuss the difference between exec and sp_executesql.
This is continuation to Part 143. Please watch Part 143 from SQL Server tutorial before proceeding.

In SQL Servere we have 2 options to execute dynamic sql
1. Exec/Execute
2. sp_executesql

We discussed sp_executesql in detail in Part 138 of SQL Server tutorial.
Please check out that video if you are new to sp_executesql.

If you do a quick search on the internet for the difference between exec and sp_executesql,
you will see that many articles on the web states using exec over sp_executesql will have the following 2 problems
1. It open doors for sql injection attacks
2. Cached query plans may not be reused and leads to poor performance

This is generally true, but if you use QUOTENAME() function you can avoid sql injection attacks and with
sql server auto-parameterisation capability the cached query plans can be reused so performance is also not an issue. Lets
understand these with examples.

What is exec() in SQL Server
Exec() or Execute() function is used to execute dynamic sql and has only one parameter i.e the dynamic sql statement you want to execute.

As you can see in the example below, we are concatenating strings to build dynamic sql statements which open doors for sql injection.

```
Declare @FN nvarchar(50)
Set @FN = 'John'
Declare @sql nvarchar(max)
Set @sql = 'Select * from Employees where FirstName = ''' +  @FN + ''''
Exec(@sql)
```

If we set @FN parameter to something like below, it drops SalesDB database

```
Declare @FN nvarchar(50)
Set @FN = ''' Drop Database SalesDB --'''
Declare @sql nvarchar(max)
Set @sql = 'Select * from Employees where FirstName = ''' + @FN + ''''
Exec(@sql)
```

However, we can prevent SQL injection using the QUOTENAME() function as shown below.

```
Declare @FN nvarchar(50)
Set @FN = ''' Drop Database SalesDB --'''
Declare @sql nvarchar(max)
Set @sql = 'Select * from Employees where FirstName = ' + QUOTENAME(@FN,'''')
--Print @sql
Exec(@sql)
```

Notice with the quotename function we are using a single quote as a delimiter.
With the use of this function if there is a single quote in the user input it is doubled.

For example, if we set @FN='John', notice the string 'John' is wrapped in single quotes

```
Declare @FN nvarchar(50)
Set @FN = 'John'
Declare @sql nvarchar(max)
Set @sql = 'Select * from Employees where FirstName = ' + QUOTENAME(@FN,'''')
Print @sql
```

When the above query is executed the following is the query printed
Select * from Employees where FirstName = 'John'

Along the same lines, if we try to inject sql, QUOTENAME() function wraps all that input in another pair of
single quotes treating it as a value for the FirstName column and prevents SQL injection.

With sql server auto-parameterisation capability the cached query plans can be reused.
SQL Server can detect parameter values and create parameterised queries on its own, even if you dont explicitly declare them. However, there are exceptions to this. Auto-parameterisation comes in 2 flavours -
Simple and Forced. We will discuss auto-parameterisation in detail in a later video.

Execute the following DBCC command to remove all entries from the plan cache
DBCC FREEPROCCACHE

Execute the following query. Notice we have set @FN='Mary'

```
Declare @FN nvarchar(50)
Set @FN = 'Mary'
Declare @sql nvarchar(max)
Set @sql = 'Select * from Employees where FirstName = ' + QUOTENAME(@FN,"")
Exec(@sql)
```

Execute the following query to retrieve what we have in the query plan cache

```
SELECT cp.usecounts, cp.cacheobjtype, cp.objtype, st.text, qp.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC
```

Notice in the 3rd row, we have an auto-parameterised query and at the moment usecounts is 1.
sql server auto parameterization

Now set @FN='Mark' and execute the same query. After the query is completed, retrieve the entries from the plan cache. Notice the usecounts for the auto-parameterised query is 2,
suggesting that the same query plan is reused.
auto parameterization sql server 2008

Along the same lines, if you change @FN='John' and execute the query, you will see that the usecounts is now 3 for the auto-parameterised query.

Summary
If you use QUOTENAME() function, you can prevent sql injection while using Exec()
Cached query plan reusability is also not an issue while using Exec(), as SQL server automatically parameterize queries.
I personally prefer using sp_executesql over exec() as we can explicitly parameterise queries instead of relying on
sql server auto-parameterisation feature or QUOTENAME() function.
I use Exec() only in throw away scripts rather than in production code.

--145-- QUOTENAME FUNCTION IN SQL SERVER

In this video we will discuss how to pass table name dynamically for stored procedure in sql server. This is one of the sql questions that is very commonly asked. Here is what we want to do.

I have a web page with a textbox as shown below. When I enter a table name in the textbox and when I click "Load Data" button,
we want to retrieve data from that respective table and display it on the page.

sql table name variable stored procedure

For the purpose of this demo, we will use the following 2 tables.

how to pass table name dynamically for stored procedure in sql server

Dynamic sql table name variable

SQL Script to create the required tables

```
Create table Countries
(
    Id int identity primary key,
    CountryName nvarchar(50)
)
Go

Insert into Countries values ('USA')
Insert into Countries values ('India')
Insert into Countries values ('UK')
Insert into Countries values ('Australia')
Insert into Countries values ('Canada')
Go

Create table Employees
(
    ID int primary key identity,
    FirstName nvarchar(50),
    LastName nvarchar(50),
    Gender nvarchar(50),
    Salary int
)
Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)
Insert into Employees values ('Steve', 'Pound', 'Male', 45000)
Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)
Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)
Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)
Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)
Insert into Employees values ('John', 'Stanmore', 'Male', 80000)
Go
```

Create the following stored procedure. Notice we are passing table name as a parameter to the stored procedure.
In the body of the stored procedure we are concatenating strings to build our dynamic sql statement.
In our previous videos we discussed that this open doors for SQL injection.

```
Create procedure spDynamicTableName
```

```
@TableName nvarchar(100)
As
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from ' + @TableName
    Execute sp_executesql @sql
End
```

So the obvious question that comes to our mind is, why are we not creating parameterised sql statement instead.
The answers is we cant. SQL Server does not allow table names and column names to be passed as parameters.
Notice in the example below, we are creating a parameterised query with @TabName as a parameter. When we execute the following code, the procedure gets created successfully.

```
Create procedure spDynamicTableName1
@TableName nvarchar(100)
As
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from @TabName'
    Execute sp_executesql @sql, N'@TabName nvarchar(100)',
    @TabName = @TableName
End
```

But when we try to execute it we get an error - Must declare the table variable "@TabName"
`Execute spDynamicTableName1 N'Countries'`

Add a Web Page to the project that we have been working with in our previous video.
Name it "DynamicTableName.aspx". Copy and paste the following HTML on the page.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Employee Search</title>
   <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
      type="text/css" />
</head>
<body style="padding-top: 10px">
   <div class="col-xs-8 col-xs-offset-2">
      <form id="form1" runat="server" class="form-horizontal">
         <div class="panel panel-primary">
            <div class="panel-heading">
               <h3>Table Lookup</h3>
            </div>
            <div class="panel-body">
               <div class="form-group">
                  <label for="inputTableName" class="control-label col-xs-4">
                     Table Name
```

```
                </label>
                <div class="col-xs-8">
                    <input type="text" runat="server" class="form-control"
                        id="inputTableName" placeholder="Please enter table name" />
                </div>
            </div>
            <div class="form-group">
                <div class="col-xs-10 col-xs-offset-2">
                    <asp:Button ID="btnLoadData" runat="server" Text="Load Data"
                        CssClass="btn btn-primary" OnClick="btnLoadData_Click" />
                    <asp:Label ID="lblError" runat="server" CssClass="text-danger">
                    </asp:Label>
                </div>
            </div>
        </div>
    </div>

    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3>Table Data</h3>
        </div>
        <div class="panel-body">
            <div class="col-xs-10">
                <asp:GridView CssClass="table table-bordered"
                    ID="gvTableData" runat="server">
                </asp:GridView>
            </div>
        </div>
    </div>
  </form>
 </div>
</body>
</html>
```

Copy and paste the following code in the code-behind page.

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

namespace DynamicSQLDemo
{
 public partial class DynamicTableName : System.Web.UI.Page
 {
  protected void btnLoadData_Click(object sender, EventArgs e)
  {
   try
   {
```

```
            if (inputTableName.Value.Trim() != "")
            {
                string strConnection = ConfigurationManager
               .ConnectionStrings["connectionStr"].ConnectionString;

                using (SqlConnection con = new SqlConnection(strConnection))
                {
                    SqlCommand cmd = new SqlCommand();
                    cmd.Connection = con;
                    cmd.CommandText = "spDynamicTableName";
                    cmd.CommandType = CommandType.StoredProcedure;

                    SqlParameter param = new
                        SqlParameter("@TableName", inputTableName.Value);
                    param.SqlDbType = SqlDbType.NVarChar;
                    param.Size = 100;
                    cmd.Parameters.Add(param);

                    con.Open();
                    SqlDataReader rdr = cmd.ExecuteReader();
                    gvTableData.DataSource = rdr;
                    gvTableData.DataBind();
                }
            }
            lblError.Text = "";
        }
        catch (Exception ex)
        {
            lblError.Text = ex.Message;
        }
    }
  }
}
```

At this point, run the application and type the following text in the "Table Name" textbox and click "Load Data" button.
Notice "SalesDB" database is dropped. Our application is prone to SQL injection as we have implemented dynamic sql in
our stored procedure by concatenating strings instead of using parameters.
Employees; Drop database SalesDB

One way to prevent SQL injection in this case is by using SQL Server built-in function - QUOTENAME(). We will discuss
QUOTENAME() function in detail in our next video. For now understand that by default, this function wraps that string
that is passed to it in a pair of brackets.
SELECT QUOTENAME('Employees') returns [Employees]

Modify the stored procedure to use QUOTENAME() function as shown below.

```
Alter procedure spDynamicTableName
@TableName nvarchar(100)
As
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from ' + QUOTENAME(@TableName)
    Execute sp_executesql @sql
End
```

At this point, type the following text in the "Table Name" textbox and click "Load Data" button. Notice you will see a message -
Invalid object name 'Employees; Drop database SalesDB'. Also "SalesDB" database is not dropped.
Employees; Drop database SalesDB

The entire text in "Table Name" textbox is wrapped in a pair of brackets by the QUOTENAME function and is treated as table name.
Since we do have a table with the specified name, we get the error - Invalid object name.

--146-- DYANMIC SQL VS STORED PROCEDURE

In this video we will discuss Quotename function in SQL Server.
This is continuation to Part 145. Please watch Part 145 from SQL tutorial before proceeding.

This function is very useful when you want to quote object names. Let us understand the use of this function with an example.

We will use the following table for the examples in this demo
quotename example in sql server

SQL Script to create and populate the table with test data

```
Create table [USA Customers]
(
    ID int primary key identity,
    FirstName nvarchar(50),
    LastName nvarchar(50),
    Gender nvarchar(50)
)
Go

Insert into [USA Customers] values ('Mark', 'Hastings', 'Male')
Insert into [USA Customers] values ('Steve', 'Pound', 'Male')
Insert into [USA Customers] values ('Ben', 'Hoskins', 'Male')
Insert into [USA Customers] values ('Philip', 'Hastings', 'Male')
Insert into [USA Customers] values ('Mary', 'Lambeth', 'Female')
Insert into [USA Customers] values ('Valarie', 'Vikings', 'Female')
Insert into [USA Customers] values ('John', 'Stanmore', 'Male')
Go
```

Let us say, we are using dynamic SQL to build our SELECT query as shown below

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers'
Set @sql = 'Select * from ' + @tableName
Execute sp_executesql @sql
```

When we execute the above script, we get the following error

```
Msg 208, Level 16, State 1, Line 1
Invalid object name 'USA'.
```

The query that our dynamic sql generates and executes is as shown below.
To see the generate SQL statement, use Print @sql.

```
Select * from USA Customers
```

Since there is a space in the table name, it has to be wrapped in brackes as shown below

```
Select * from [USA Customers]
```

One way to fix this is by including the brackets in @tableName variable as shown below

```
Set @tableName = '[USA Customers]'
```

The other way to fix this is by including the brackets in @sql variable as shown below

```
Set @sql = 'Select * from [' + @tableName +']'
```

While both of the above methods give the result we want,
it is extremely dangerous because it open doors for sql injection.

If we set the brackets in @tableName variable,
sql can be injected as shown below and SalesDB database is dropped

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = '[USA Customers] Drop Database SalesDB'
Set @sql = 'Select * from ' + @tableName
Execute sp_executesql @sql
```

If we set the brackets in @sql variable, sql can be injected as
shown below and SalesDB database is dropped

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers] Drop Database SalesDB --'
Set @sql = 'Select * from [' + @tableName +']'
```

```
Execute sp_executesql @sql
```

So, the right way to do this is by using QUOTENAME() function as shown below.

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers Drop Database SalesDB --'
Set @sql = 'Select * from ' + QUOTENAME(@tableName)
Execute sp_executesql @sql
```

When we execute the above script we get the following error.
SalesDB database is not dropped. The reason we get this error is because we do not have a table with name -
[USA Customers Drop Database SalesDB --]. To see the sql statement use PRINT @sql.
Invalid object name 'USA Customers Drop Database SalesDB --'.

If we set @tableName = 'USA Customers', the query executes successfully, without the threat of SQL injection.

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers'
Set @sql = 'Select * from ' + QUOTENAME(@tableName)
Execute sp_executesql @sql
```

If you want to use sql server schema name "dbo" along with the table name,
then you should not use QUOTENAME function as shown below.

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'dbo.USA Customers'
Set @sql = 'Select * from ' + QUOTENAME(@tableName)
Execute sp_executesql @sql
```

The above query produces the following error
Invalid object name 'dbo.USA Customers'

Instead use QUOTENAME function as shown below

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers'
```

```
Set @sql = 'Select * from ' + QUOTENAME('dbo') + '.' + QUOTENAME(@tableName)
Execute sp_executesql @sql
```

QUOTENAME() function
Takes two parameters - the first is a string, and the second is a delimiter that you want SQL server to use to wrap the string in.
The delimiter can be a left or right bracket ([]), a single quotation mark (') ' , or a double quotation mark (") "
The default for the second parameter is []
QUOTENAME() function examples

SELECT QUOTENAME('USA Customers','"') returns "USA Customers"

SELECT QUOTENAME('USA Customers','"') returns 'USA Customers'

All the following statements return [USA Customers]
SELECT QUOTENAME('USA Customers')
SELECT QUOTENAME('USA Customers','[')
SELECT QUOTENAME('USA Customers',']')

If you use a delimiter other than a single quotation mark, double quotation mark, left bracket or a right bracket, you get NULL.
The following statement returns NULL.

SELECT QUOTENAME('USA Customers','*')

For some reason if you have a bracket in the table name, QUOTENAME function will double it to indicate an escape character.
SELECT QUOTENAME('USA] Customers') returns [USA]] Customers]

To remove the QUOTENAME use, PARSENAME() function as shown below.

```
Declare @tableName nvarchar(50)
Set @tableName = 'USA ] Customers'
Set @tableName = QUOTENAME(@tableName)
Print @tableName

Set @tableName = PARSENAME(@tableName,1)
Print @tableName
```

Result:
[USA]] Customers]
USA] Customers

PARSENAME() takes 2 parameters. The first is the object name and the second is the object piece. It is an int and can be
1 = Object name
2 = Schema name
3 = Database name

4 = Server name

--147-- DYNAMIC SQL OUTPUT PARAMETER

Dynamic sql output parameter
Suggested Videos
Part 145 - Dynamic sql table name variable
Part 146 - Quotename function in SQL Server
Part 147 - Dynamic SQL vs Stored Procedure

In this video we will discuss, how to use output parameters with dynamic sql. Let us understand this with an example.

We will use the following Employees table in this demo.
Dynamic sql output parameter

SQL script to create Employees table

```
Create table Employees
(
    ID int primary key identity,
    FirstName nvarchar(50),
    LastName nvarchar(50),
    Gender nvarchar(50),
    Salary int
)
Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)
Insert into Employees values ('Steve', 'Pound', 'Male', 45000)
Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)
Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)
Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)
Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)
Insert into Employees values ('John', 'Stanmore', 'Male', 80000)
Go
```

We want to write a dynamic sql statement that returns total number of male of female employees. If the gender value is specified as "Male", then the query should return total male employees. Along the same lines, if the the value for gender is "Female", then we should get total number of female employees.

The following dynamic sql, will give us what we want. In this case, the query returns total number of "Male" employees.
If you want the total number of female employees, simply set @gender='Female'.

```
Declare @sql nvarchar(max)
Declare @gender nvarchar(10)
Set @gender = 'Male'
Set @sql = 'Select Count(*) from Employees where Gender=@gender'
Execute sp_executesql @sql, N'@gender nvarchar(10)', @gender
```

At the moment we are not using output parameters. If you want the count of employees to be returned using an OUTPUT parameter,
then we have to do a slight modification to the query as shown below.
The key here is to use the OUTPUT keyword in your dynamic sql.
This is very similar to using OUTPUT parameters with a stored procedure.

```
Declare @sql nvarchar(max)
Declare @gender nvarchar(10)
Declare @count int
Set @gender = 'Male'
Set @sql = 'Select @count = Count(*) from Employees where Gender=@gender'
Execute sp_executesql @sql, N'@gender nvarchar(10), @count int OUTPUT',
                            @gender, @count OUTPUT
Select @count
```

The OUTPUT parameter returns NULL, if you forget to use OUTPUT keyword.
The following query returns NULL, as we removed the OUTPUT keyword from @count parameter

```
Declare @sql nvarchar(max)
Declare @gender nvarchar(10)
Declare @count int
Set @gender = 'Male'
Set @sql = 'Select @count = Count(*) from Employees where Gender=@gender'
Execute sp_executesql @sql, N'@gender nvarchar(10), @count int OUTPUT',
                            @gender, @count
Select @count
```

--148-- TEMP TABLES IN DYNAMIC SQL

In this video we will discuss the implications of creating temp tables in dynamic sql

Temp tables created by dynamic SQL are not accessible from the calling procedure. They are dropped when the dynamic SQL block
in the stored procedure completes execution.

Let us understand this with an example. Notice in the example below, all the following 3 operations are in the block of dynamic sql code.
1. Creating the Temp Table
2. Populating the Temp Table
3. Select query on the Temp Table

```
Create procedure spTempTableInDynamicSQL
as
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Create Table #Test(Id int)
                insert into #Test values (101)
                Select * from #Test'
    Execute sp_executesql @sql
End
```

So when we execute the above procedure we are able to access data from the Temp Table.
Execute spTempTableInDynamicSQL

Now, lets move the SELECT statement outside of the dynamic sql code
block as shown below and ALTER the stored procedure.

```
Alter procedure spTempTableInDynamicSQL
as
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Create Table #Test(Id int)
                insert into #Test values (101)'
    Execute sp_executesql @sql
    Select * from #Test
End
```

At this point, execute the stored procedure. Notice, we get the error - Invalid object name '#Test'.
This is because temp tables created by dynamic SQL are not accessible from the calling procedure.
They are dropped when the dynamic SQL block in the stored procedure completes execution.
Execute spTempTableInDynamicSQL

On the other hand, dynamic SQL block can access temp tables created by the calling stored procedure.
Lets
prove this by modifying the stored procedure as shown below.

```
Alter procedure spTempTableInDynamicSQL
as
Begin
    Create Table #Test(Id int)
    insert into #Test values (101)
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from #Test'
    Execute sp_executesql @sql
End
```

At this point, execute the stored procedure. Notice that we are able to access the temp table,
which proves that dynamic SQL block can access temp tables created by the calling stored procedure.

```
Execute spTempTableInDynamicSQL
```

Summary
Temp tables created by dynamic SQL are not accessible from the calling procedure.
They are dropped when the dynamic SQL block in the stored procedure completes execution.
On the other hand, dynamic SQL block can access temp tables created by the calling stored procedure.