POC on

Enhancements in SQL Server Denali CTP3

Author: ASP - Data Architecture Consulting

Created: 2011-08-11

1 T-SQL ENHANCEMENTS 14

1.1 INTRODUCTION 14

1.2 CONVERSION FUNCTIONS 14
1.2.1 PARSE 14
1.2.2 TRY_PARSE 15
1.2.3 TRY_CONVERT 15

1.3 Date and Time Functions 16
1.3.1 DATEFROMPARTS 16
1.3.2 DATETIME2FROMPARTS 16
1.3.3 DATETIMEFROMPARTS 17
1.3.4 DATETIMEOFFSETFROMPARTS 17
1.3.5 EOMONTH 18
1.3.6 SMALLDATETIMEFROMPARTS 18
1.3.7 TIMEFROMPARTS 19

1.4 Logical functions 19
1.4.1 CHOOSE 19
1.4.2 IIF 20

1.5 String functions 21
1.5.1 CONCAT 21
1.5.2 FORMAT 21

1.6 Analytic Functions 22
1.6.1 CUME_DIST 22
1.6.2 FIRST_VALUE 23
1.6.3 LAG 24
1.6.4 LAST_VALUE 25
1.6.5 LEAD 26
1.6.6 PERCENTILE_CONT 26
1.6.7 PERCENTILE_DISC 27
1.6.8 PERCENT_RANK 28

2 SEQUENCE 29

2.1 Introduction 29

2.2 Syntax 29
2.2.1 CREATE SEQUENCE 29
2.2.2 ALTER SEQUENCE 30
2.2.3 DROP SEQUENCE 30

2.2.4 Other system Functions and Objects related to Sequence 30
2.2.4.1 NEXT VALUE FOR 30
2.2.4.2 sys.sequence 31
2.2.4.3 sp_sequence_get_range 31

2.3 Important Architectural Consideration and Limitations 31

2.4 How does it differ from Identity Column? 32

2.5 Performance Comparison of Sequence and Identity 32

2.6 How to create Sequence from Management Studio 33
2.6.1 Sequences comes under “Programmability” 33
2.6.2 New Sequence window 34
2.6.3 Data type to be chosen carefully 35

2.7 Different Scenario of Sequence Usage in SQL Server 36

3 EXCEPTION HANDLING ENHANCEMENT IN SQL SERVER 2011 39

3.1 THROW statement 39

3.2 Comparisons between Try…Catch model and Throw 41

3.3 XACT_ABORT IN THROW 49

3.4 Points to remember 52

4 CHANGE STARTUP PARAMETER 53

4.1 List of Startup Options 53

4.2 Compatibility Support 55

4.3 How to Change Startup Parameter in SQL Server Denali 55

5 CODE SNIPPETS 59

5.1 Completing the Snippet Statement 59

5.2 Inserting Code Snippets 59

5.3 Insert Surround-with Transact-SQL snippets 63
5.3.1 To insert a surround-with snippet 63

5.3.2 BEGIN, WHILE and IF Blocks 63

5.4 Snippet Management 64

6 JUNEAU 66

6.1 Features of JUNEAU: 66
6.1.1 Model-Based Development 66
6.1.2 SSMS in Server Explorer 66
6.1.3 Offline Development in Visual Studio 67
6.1.4 Versioning and Snapshots 68
6.1.5 Targeting SQL Azure 68

7 OVER CLAUSE SUPPORT ENHANCED 69

7.1 Using the OVER clause with the ROW_NUMBER function 70

7.2 Using the OVER clause with aggregate functions 71

7.3 Using the OVER clause with an aggregate function in a calculated value 71
1.1.1 71

7.4 Using the AVG and SUM functions with the OVER clause to provide a moving average and cumulative total 72

7.5 Specifying the ROWS clause 72

7.6 Limitations and Restrictions 73

8 DYNAMIC MANAGEMENT VIEWS AND FUNCTIONS 74

8.1 sys.dm_exec_query_stats (Transact-SQL) 74

8.2 sys.dm_os_volume_stats (Transact-SQL) 74

8.3 sys.dm_os_windows_info (Transact-SQL) 74

8.4 sys.dm_server_memory_dumps (Transact-SQL) 74

8.5 sys.dm_server_services (Transact-SQL) 74

8.6 sys.dm_server_registry (Transact-SQL) 74

9 AD-HOC QUERY PAGING 75

9.1 Using OFFSET and FETCH to limit the rows returned 76
E. Running multiple queries in a single transaction 78

10 CONTAINED DATABASES 80

10.1 Problems with non contained databases 80
10.1.1 When a database is moved from one to another SQL Server instance, instance specific information such as logins and agent job cannot be moved. These tasks need to be recreated again. 80
10.1.2 Application deployment in a server from the development server may cause issues due to environmental mismatch. 80
10.1.3 Login and job information are available at instance level. So, users need to be granted permission at the instance level causing security issue. 80
10.1.4 It is difficult to maintain and administer a single database independently. 80

10.2 Introduction to Contained database 80
10.2.1 This is a self-contained database i.e. it includes all the database settings and required metadata information. 80
10.2.2 This is instance or server independent. Users can be authenticated in the database itself. 80
10.2.3 This database can be moved easily to another instance or server as there is no external dependency. 80
10.2.4 Tempdb collation issues are resolved with contained database. Temporary objects are created as per the collation setting of the database, not of tempdb. 80

10.3 Contained database terms: 80
10.3.1 Application boundary 80
10.3.2 Application model 80
10.3.3 Contained 80
10.3.4 Uncontained 80
10.3.5 Non-contained database 81
10.3.6 Fully contained database 81
10.3.7 Partially contained database 81
10.3.8 Contained user 81
10.3.8.1 Contained database users with password which are authenticated by the database 81
10.3.8.2 Windows principals that can directly connect to the database. Master database logins are not required for these users. 81

10.4 Contained database creation steps 81
10.4.1 Enable contained database 81
10.4.1.1 SQL Code 81
10.4.1.2 Through SSMS 81
10.4.2 Create/Alter contained database 82
10.4.2.1 SQL Code 82
10.4.2.2 Through SSMS 83
10.4.3 Create user in contained database 83

10.4.3.1 SQL Code 83
10.4.3.2 Through SSMS 83
10.4.4 Login to the contained database 84
10.4.5 Uncontaining the database 85

10.5 Limitations 85

11 COLUMN-STORE INDEXES (CSI) 86

11.1 Introduction to column-stores 86

11.2 Column-store advantages 87
11.2.1 Less data during query processing 87
11.2.2 Efficient compression 87

11.3 Column-store index (CSI) in SQL Server 88
11.3.1 Batch mode execution 88
11.3.2 Points to be considered before using CSI 88
11.3.2.1 Separate data structure 88
11.3.2.2 Only one CSI per table 88
11.3.2.3 No DML operations 88
11.3.2.4 Not all data types are supported 88
11.3.3 Metadata 89

12 ENHANCED COLLATION SUPPORT 90

12.1 Supplementary Characters 90

12.2 Limitations of Supplementary Characters 90

13 DATABASE ENGINE TUNING ADVISOR ENHANCEMENTS 92

13.1 To tune a database by using the plan cache 92

14 INSTALLATION ENHANCEMENTS 93

14.1 Installing Prerequisites during SQL Server Code-Named "Denali" Setup 93

14.2 Changes to Operating System Requirements 93

14.3 Data Quality Services 94

14.4 Product Update 94

14.5 Server Core Installation **94**

14.6 SQL Server multi-subnet clustering **94**

14.7 SMB file share is a supported storage option **94**

15 MULTI MONITOR SUPPORT **95**

16 ZOOM **96**

17 REPLICATION ENHANCEMENT **97**

17.1 Replication Backward Compatibility **97**
17.1.1 97
17.1.3 Deprecated Features in SQL Server Replication 97
17.1.2 Behavior Changes in SQL Server Replication 97
17.1.3 Breaking Changes in SQL Server Replication 97

18 TASK LIST **98**

19 SSIS ENHANCEMENTS **99**

19.1 Troubleshooting Performance and Data Issues **99**

19.2 Troubleshooting Capability **99**
19.2.1 Get performance statistics and other information for an execution 99

RELATED VIEWS, PROCEDURES, AND FUNCTIONS **99**
19.2.1.1 catalog.executions (SSISDB Database) 99
19.2.1.2 catalog.execution_component_phases 99
19.2.1.3 dm_execution_performance_counters (SSISDB Database) 99
19.2.2 Add, remove, and query data taps in a package data flow 100
19.2.2.1 catalog.add_data_tap 100
19.2.2.2 catalog.remove_data_tap 101
19.2.2.3 catalog.execution_data_taps 101
19.2.2.4 catalog.execution_data_statistics 101
19.2.3 Create a dump for a running package 101
19.2.3.1 catalog.create_execution_dump 101
19.2.3.2 catalog.set_execution_parameter_value (SSISDB Database) 102

20 SECURITY ENHANCEMENTS **103**

20.1 Provisioning During Setup 103

20.2 New Permissions 103

20.3 New Role Management 103

20.4 ALTER ROLE (Transact-SQL) 103

20.5 IS_ROLEMEMBER (Transact-SQL) 104

20.6 Default Schema for Groups 104

20.7 SQL Server Audit Enhancements 104

20.8 Database Engine Access is allowed Through Contained Databases 105

20.9 Hashing Algorithms 105

20.10 Further Deprecation of RC4 105

20.11 Certificate Key Length 105

20.12 Service Master Key and Database Master Key Encryption changes from 3DES to AES 106

20.13 Certificates can be Created from Binary 106

21 AVAILABILITY ENHANCEMENTS 107

21.1 Database Mirroring's Drawbacks 107

21.2 High Availability Solutions 108

21.3 AlwaysOn SQL Server Failover Cluster Instances 111

21.4 AlwaysOn Availability Groups 111

21.5 Indirect Checkpoints 112

21.6 How to Implement HADR (a high-availability and disaster recovery solution) 113

22 ALWAYSON AVAILABILITY GROUPS 114

22.1 Overview of AlwaysOn Availability Groups 114

22.1.1 Availability Replicas and Roles 114

22.1.2 Availability Modes 115

22.1.3 Types of Failover 116
22.1.4 Allowing Read-Only Access to Secondary replicas 117
22.1.5 Client Connections 117
22.1.6 Automatic Page Repair 118
22.1.7 Interoperability and Coexistence with Other Database Engine Features 118

23 FULL-TEXT SEARCH 119

23.1 Property Search 119
23.1.1 Known Restrictions on Property Searching 119
23.1.2 Search Property Lists 119
23.1.3 Configuring a Full-Text Index for Property Searching 120
23.1.4 Creating a Search Property List 120
23.1.4.1 To create a search property list in Management Studio 120
23.1.4.2 Viewing and Changing a Search Property List 120
23.1.4.3 Deleting a Search Property List 121

23.2 Customizable NEAR 121
23.2.1 The Custom Proximity Term 122
23.2.2 How Maximum Distance Is Measured 123
23.2.3 Combining a Custom Proximity Term with Other Terms 123

23.3 New Word Breakers and Stemmers 124

24 SPATIAL DATA TYPE ENHANCEMENT 125

24.1 Circular Arc Segment Support for Spatial Types 125
24.1.1 CircularString instances 125
24.1.2 CompoundCurve instances 126
24.1.3 CurvePolygon 128
Accepted instances 128
Valid instances 128
Geometry data type 128
Geography data type 129
A. Instantiating a Geometry Instance with an Empty CurvePolygon 129

24.2 New methods for geometry and geography data types 129
24.2.1 OGC Methods on Geography Instances 129
24.2.2 Extended Methods on Geography Instances 130
24.2.3 OGC Methods on Geometry Instances 130
24.2.4 Extended Methods on Geometry Instances 132
24.2.5 SQL MM Methods on Geography Instances 132

24.3 New static aggregate methods for geometry data type and geography data type 132
24.3.1 Extended Static Geography Methods 133
24.3.2 Extended Static Geometry Methods 133

25 SERVICE BROKER 134

25.1 New Conversation Priorities 134
25.1.1 Conversation Priorities 134
25.1.2 How Service Broker Assigns Priority Levels 134

25.2 New Diagnostic Utility 136
25.2.1 ssbdiagnose Utility 136

25.3 New Service Broker Elements in Object Explorer 136

25.4 New System Monitor Object and Counters 136

25.5 New Service Broker Tutorials 137
25.5.1 Service Broker Tutorials 137

26 BUSINESS INTELLIGENCE 139
To drag and drop text 139

26.1 Based on Tabular Models 139

26.2 Coexists with Report Builder 140

26.3 Author Tabular Models in Business Intelligence Development Studio 140
26.3.1 Tabular Modeling (SSAS) 140
26.3.2 Tabular Model Solutions (SSAS) 140

26.4 Tabular Modeling (Adventure Works Tutorial) 141

REFER: **TABULAR MODELING (ADVENTURE WORKS TUTORIAL) 142**

27 ONLINE INDEX REBUILD 142

27.1 Online Index Operations 142

27.2 How Online Index Operations Work 143
27.2.1 Online Index Structures 143
27.2.2 Online Index Activities 144
27.2.3 Source Structure Activities 145

27.2.4 Target Structure Activities 147

27.3 Guidelines for Online Index Operations 148
27.3.1 Disk Space Considerations 150
27.3.2 Performance Considerations 150
27.3.3 Transaction Log Considerations 151

28 IT ADMINISTRATION 152

28.1 Provisioning During Setup 152

28.2 New Permissions 152

28.3 New Role Management 152
28.3.1 Create Server Role (Transact-SQL) 152
28.3.2 ALTER SERVER ROLE (Transact-SQL) 154
28.3.3 Fixed server roles 154
28.3.4 User-defined server roles 154

28.4 DROP SERVER ROLE (Transact-SQL) 156

28.5 ALTER ROLE (Transact-SQL) 157

28.6 IS_ROLEMEMBER (Transact-SQL) 158

28.7 SQL Server Audit Enhancements 158
28.7.1 CREATE SERVER AUDIT (Transact-SQL) 160
28.7.2 ALTER SERVER AUDIT (Transact-SQL) 163

29 BEYOND RELATIONAL 168

30 METADATA DISCOVERY 169

30.1 bcp functions 169
30.1.1 bcp_columns 169
30.1.2 bcp_control 170
30.1.3 BCPABORT 170
30.1.4 BCPBATCH 170
30.1.5 BCPDELAYREADFMT 171
30.1.6 BCPFILECP 171
30.1.7 BCPFILEFMT 171
30.1.8 BCPFIRST 171
30.1.9 BCPFIRSTEX 171

30.1.10 BCPFMTXML 172
30.1.11 BCPHINTS 172
30.1.12 BCPKEEPIDENTITY 172
30.1.13 BCPKEEPNULLS 172
30.1.14 BCPLAST 172
30.1.15 BCPLASTEX 173
30.1.16 BCPMAXERRS 173
30.1.17 BCPODBC 173
30.1.18 BCPROWCOUNT 173
30.1.19 BCPTEXTFILE 173
30.1.20 BCPUNICODEFILE 173
30.1.21 bcp_getcolfmt 174
30.1.22 bcp_readfmt 175
30.1.23 bcp_setcolfmt 175

30.2 ODBC functions 180
30.2.1 SQLNumResultCols 180
30.2.2 SQLDescribeCol 180
30.2.3 SQLDescribeCol Support for Enhanced Date and Time Features 181
30.2.4 SQLNumParams 181
30.2.5 SQLDescribeParam 182
30.2.6 SQLDescribeParam and Table-Valued Parameters 183
30.2.7 SQLDescribeParam Support for Enhanced Date and Time Features 183

31 FILESTREAM SUPPORT 185

31.1 FILESTREAM Support (OLE DB) 185

31.2 FILESTREAM Support (ODBC) 185

31.3 Access FILESTREAM Data with OpenSqlFilestream 186

31.4 Querying for FILESTREAM Columns 189

31.5 Down-Level Compatibility **190**

32 FILE TABLES 190
32.1.1 Requirements and Restrictions for Creating a FileTable 192
32.1.2 Changing the Directory for a FileTable 193
32.1.3 Requirements and Restrictions for Altering a FileTable 193
32.1.4 196
- Loading Files into a FileTable 196
- Bulk Load Files into a FileTable 196

32.1.5 Identify the Locks Held by File Tables 198
- Partitioning and File Tables 199
- Replication and File Tables 199
- Transaction Semantics and File Tables 199
- Query Notifications and File Tables 200
- SELECT INTO and File Tables 200
- Triggers and File Tables 200
- Views and File Tables 200
- Snapshot Isolation and File Tables 201
- Contained Databases and File Tables 201
- DBCC and File Tables 201

33 REFERENCES 204

1 T-SQL Enhancements

1.1 INTRODUCTION

The new version of Microsoft SQL Server, code-named "Denali" is in its Community Technology Preview 3 (CTP 3) when this document was created. It has 14 new built-in functions. These functions ease the path of migration for information workers by emulating functionality that is found in the expression languages of many desktop applications. Not only that, these functions will also be useful to experienced users of SQL Server.

1.2 CONVERSION FUNCTIONS

1.2.1 PARSE

PARSE returns the result of an expression, translated to specified data type if the translation is possible; otherwise, it raises an error.

Syntax

PARSE (string_value AS data_type [USING culture]) string_value

- String_value – string value to parse into the specified data type.
- Data_type – return data type, numeric or date time type
- Culture - a language (English, Japanese, Spanish, Danish, French etc.) which will used by SQL Server to interpret data.

Handling NULL values

1. If a null constant is passed, an error is raised. A null value cannot be parsed into a different data type in a culturally aware manner.
2. If a parameter with a null value is passed at run time, then a null is returned, to avoid canceling the whole batch.

Sample code

PARSE into datetime2

```
SELECT PARSE ('Monday, 13 December 2010' AS datetime2 USING 'en-US')
AS Result
```

PARSE with currency symbol

```
SELECT PARSE ('€34598' AS money USING 'de-DE') AS Result
```

PARSE with implicit setting of language

```
SET LANGUAGE 'English'
SELECT PARSE ('12/16/2010' AS datetime2) AS Result
```

1.2.2 TRY_PARSE

TRY_PARSE tries to translate the result of an expression to specified data type if the translation is possible, otherwise, it returns NULL**.**

Syntax

TRY_PARSE (string_value AS data_type [USING culture])

- CULTURE, means a language (English, Japanese, Spanish, Danish, French etc.) which will be used by SQL Server to interpret data.

Sample code

Detecting nulls with TRY_PARSE

```
SELECT
    CASE WHEN TRY_PARSE('Aragorn' AS decimal USING 'sr-Latn-CS') IS NULL
        THEN 'True'
        ELSE 'False'
END
AS Result
```

Using IIF with TRY_PARSE and implicit culture setting

```
SET LANGUAGE English
SELECT IIF(TRY_PARSE('01/01/2011' AS datetime2) IS NULL, 'True', 'False') AS
Result
```

1.2.3 TRY_CONVERT

TRY_CONVERT returns a value cast to the specified data type if the cast succeeds; otherwise, returns null.

Syntax

TRY_CONVERT (data_type [(length)], expression [, style])

- Data_type [(length)] is the data type into which to cast expression.
- Expression is the value to be cast.
- Style is optional integer expression that specifies how the TRY_CONVERT function is to translate expression.

Sample code

TRY_CONVERT returns null

```
SELECT
    CASE WHEN TRY_CONVERT(float,'test') IS NULL
```

```
    THEN 'Cast failed'
    ELSE 'Cast succeeded'
END AS Result

TRY_CONVERT succeeds
SELECT TRY_CONVERT (datetime2, '12/31/2010') AS Result
```

1.3 Date and Time Functions

1.3.1 DATEFROMPARTS

DATEFROMPARTS returns a date value for the specified year, month, and day.

Syntax

DATEFROMPARTS (year, month, day)

- Year is the integer expression specifying a year.
- Month is the integer expression specifying a month, from 1 to 12.
- Day is the integer expression specifying a day.

Sample code

```
SELECT DATEFROMPARTS (2010, 12, 31) AS Result;
```

1.3.2 DATETIME2FROMPARTS

DATETIME2FROMPARTS returns a datetime2 value for the specified date and time and with the specified precision.

Syntax

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

- Year is the Integer expression specifying a year.
- Month is the Integer expression specifying a month.
- Day is the integer expression specifying a day. Hour is Integer expression specifying hours.
- Minute is Integer expression specifying minutes.
- Seconds is the integer expression specifying seconds.
- Fraction is the integer expression specifying fractions.
- Precision is integer literal specifying the precision of the datetime2 value to be returned.

Note:
If the arguments are not valid, an error is raised.
If required arguments are null, then null is returned.
If the precision argument is null, then an error is raised.

Sample code

```
SELECT DATETIME2FROMPARTS ( 2010, 12, 31, 23, 59, 59, 0, 0 ) AS Result;
```

1.3.3 DATETIMEFROMPARTS

DATETIMEFROMPARTS returns a date time value for the specified date and time.

Syntax

DATETIMEFROMPARTS (year, month, day, hour, minute, seconds, milliseconds)

- Returns a fully initialized DATETIME value.
- If the arguments are not valid, then an error is raised.
- If required arguments are null, then a null is returned.

Sample code

```
SELECT DATETIME2FROMPARTS ( 2010, 12, 31, 23, 59, 59, 0, 0 ) AS Result;
```

1.3.4 DATETIMEOFFSETFROMPARTS

DATETIMEOFFSETFROMPARTS returns a DATETIMEOFFSET value for the specified date and time and with the specified offsets and precision.

Syntax

DATETIMEOFFSETFROMPARTS (year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset, precision)

- The offset arguments are used to represent the time zone offset. If the offset arguments are omitted, then the time zone offset is assumed to be 00:00, that is, there is no time zone offset.
- If the offset arguments are specified, then both arguments must be present and both must be positive or negative.
- If minute_offset is specified without hour_offset, an error is raised.
- If other arguments are not valid, then an error is raised.
- If required arguments are null, then a null is returned. However, if the precision argument is null, then an error is raised.

Sample code

```
SELECT DATETIMEOFFSETFROMPARTS (2010, 12, 31, 14, 23, 23, 0, 12, 0, 7)
AS Result;
```

1.3.5 EOMONTH

EOMONTH returns the last day of the month that contains the specified date, with an optional offset.

Syntax

EOMONTH (start_date [, month_to_add])

- start_date is the date expression specifying the date for which to return the last day of the month.
- month_to_add is the optional integer expression specifying the number of months to add to start_date.

EOMONTH takes the date specified by start_date and returns the date value that represents the last day of the same month. If required arguments are null, then a null is returned.

Sample code

EOMONTH with explicit datetime type

```
DECLARE @date DATETIME;
SET @date = '12/31/2010';
SELECT EOMONTH ( @date ) AS Result;
```

EOMONTH with string parameter and implicit conversion

```
DECLARE @date VARCHAR(255);
SET @date = '12/31/2010';
SELECT EOMONTH ( @date ) AS Result;
```

1.3.6 SMALLDATETIMEFROMPARTS

SMALLDATETIMEFROMPARTS () returns a smalldatetime value for the specified date and time.

Syntax

SMALLDATETIMEFROMPARTS (year, month, day, hour, minute)

- If the arguments are not valid, then an error is thrown. If required arguments are null, then null is returned.

Sample code

```
SELECT SMALLDATETIMEFROMPARTS ( 2010, 12, 31, 23, 59 ) AS Result
```

1.3.7 TIMEFROMPARTS

TIMEFROMPARTS () returns a time value for the specified time and with the specified precision.

Syntax

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

- TIMEFROMPARTS returns a fully initialized time value.
- If the arguments are invalid, then an error is raised.
- If any of the parameters are null, null is returned. However, if the precision argument is null, then an error is raised.

Sample code

```
SELECT TIMEFROMPARTS ( 23, 59, 59, 0, 0 ) AS Result;
```

1.4 Logical functions

1.4.1 CHOOSE

CHOOSE returns the item at the specified index from a list of values.

Syntax

CHOOSE (index, val_1, val_2 [, val_n])

- "index" is an integer expression that represents a 1-based index into the list of the items following it. If the provided index value has a numeric data type other than int, then the value is implicitly converted to an integer.
- If the index value exceeds the bounds of the array of values, then CHOOSE returns null.
- val_1 … val_n are list of values of any data type.

CHOOSE acts like an index into an array, where the array is composed of the arguments that follow the index argument. The index argument determines which of the following values will be returned.

Sample code

```
The following example returns the third item from the list of values that
```

```
is provided.

SELECT CHOOSE ( 3, 'Manager', 'Director', 'Developer', 'Tester' ) AS
Result;
```

1.4.2 IIF

IIF returns one of two values, depending on whether the Boolean expression evaluates to true or false.

Syntax

IIF (boolean_expression, true_value, false_value)

- Boolean_expression is a valid Boolean expression. If this argument is not a Boolean expression, then a syntax error is raised.
- True_value is the value to be returned if boolean_expression evaluates to true.
- False_value is the value to be returned if boolean_expression evaluates to false.

Returns the data type with the highest precedence from the types in true_value and false_value.

IIF is a shorthand way for writing a CASE statement. It evaluates the Boolean expression passed as the first argument, and then returns either of the other two arguments based on the result of the evaluation. That is, the true_value is returned if the Boolean expression is true, and the false_value is returned if the Boolean expression is false or unknown.

Sample code

Simple IIF example

```
DECLARE @a int = 45;
DECLARE @b int = 40;
SELECT IIF ( @a > @b, 'TRUE', 'FALSE' ) AS Result;
```

IIF with NULL constants

```
SELECT IIF ( 45 > 30, NULL, NULL ) AS Result;

The result of this statement is an error.

Msg 8133, Level 16, State 1, Line 1
At least one of the result expressions in a CASE specification must be an
expression other than the NULL constant.
```

IIF with NULL parameters

```
DECLARE @P INT = NULL;
DECLARE @S INT = NULL;
```

```
SELECT IIF ( 45 > 30, @p, @s ) AS Result;
```

1.5 String functions

1.5.1 CONCAT

CONCAT returns a string that is the result of concatenating two or more string values.

Syntax

CONCAT (string_value1, string_value2 [, string_valueN])

- CONCAT takes a variable number of string arguments and concatenates them into a single string.
- It requires a minimum of two input values; otherwise, an error is raised.
- All arguments are implicitly converted to string types and then concatenated.
- Null values are implicitly converted to an empty string.
- If all the arguments are null, then an empty string of type varchar (1) is returned.

Sample code

Using CONCAT

```
SELECT CONCAT ( 'Happy', 'Christmas', 12, '/', '25' ) AS Result;
```

Using CONCAT with NULL values

```
CREATE TABLE #temp (
    Emp_name nvarchar (200) NOT NULL,
    Emp_middlename nvarchar (200) NULL,
    Emp_lastname nvarchar (200) NOT NULL);
INSERT INTO #temp VALUES ( 'Name', NULL, 'Lastname' );
SELECT CONCAT (emp_name, emp_middlename, emp_lastname) FROM #temp AS Result;
```

1.5.2 FORMAT

FORMAT returns a value formatted with the specified format and optional culture.

Syntax

FORMAT (value, format [, culture])

- Culture is optional nvarchar argument specifying a culture.
- If the culture argument is not provided, then the language of the current session is used. This language is set either implicitly or explicitly by using the SET LANGUAGE statement.

- Culture accepts any culture supported by the .NET Framework as an argument; it is not limited to the languages explicitly supported by SQL Server.
- If the culture argument is not valid, FORMAT raises an error.
- The length of the return value is determined by the format.
- In the case of errors other than a culture that is not valid, such as a format that is not valid, FORMAT returns a null.

Sample code

Simple FORMAT example

```
DECLARE @d DATETIME = '01/01/2011';
SELECT FORMAT ( @d, 'd', 'en-US' ) AS Result;
```

FORMAT with custom formatting strings

```
DECLARE @d DATETIME = GETDATE();
SELECT FORMAT( @d, 'dd/MM/yyyy' ,'en-US' ) AS Result;
```

1.6 Analytic Functions

1.6.1 CUME_DIST

CUME_DIST computes the relative position of a specified value in a group of values. For a row *r*, assuming ascending ordering, the CUME_DIST of *r* is the number of rows with values lower than or equal to the value of *r*, divided by the number of rows evaluated in the partition or query result set. CUME_DIST is similar to the PERCENT_RANK function.

Syntax

CUME_DIST( ) OVER ([partition_by_clause] order_by_clause)

- OVER ([partition_by_clause] order_by_clause) divides the result set produced by the FROM clause into partitions to which the function is applied.
- If not specified, the function treats all rows of the query result set as a single group.
- order_by_clause determines the logical order in which the operation is performed.
- The <rows or range clause> of the OVER syntax cannot be specified in a CUME_DIST function.

The range of values returned by CUME_DIST is greater than 0 and less than or equal to 1. NULL values are included by default and are treated as the lowest possible values.

Sample code

```
CREATE TABLE [Results] (
```

```
[Subject] varchar(10),
[Student] varchar(50),
[Marks] int)
-- Inserting sample records
INSERT INTO [Results]
VALUES ('Maths','Student1',45), ('Physics','Student2',45),
('Physics','Student1',50), ('Chemistry','Student3',20),
('Physics','Student3',35), ('Biology','Student1',20),
('Biology','Student2',60), ('Biology','Student3',65),
('Chemistry','Student1',75), ('Biology','Student4',30)
GO
-- Querying with  CUME_DIST()
SELECT  [Student], [Subject], [Marks],
CUME_DIST() OVER(PARTITION BY [Subject] ORDER BY [Marks])
FROM [Results]
ORDER BY [Subject],[Marks]
GO
```

1.6.2 FIRST_VALUE

FIRST_VALUE returns the first value in an ordered set of values.

Syntax

FIRST_VALUE ([scalar_expression) OVER ([partition_by_clause] order_by_clause [rows_range_clause])

- Scalar_expression is the value to be returned. Scalar_expression can be a column, subquery, or other arbitrary expression that results in a single value.
- Partition_by_clause divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group.
- Order_by_clause determines the logical order in which the operation is performed.
- Rows_range_clause further limits the rows within the partition by specifying start and end points.

Sample code

```
CREATE TABLE TESTCTP3 (
Objectid int,
ObjectName varchar(500),
```

```
ObjectType varchar(100) ) ;

-- Inserting sample records
INSERT INTO testctp3
VALUES
(1,'sp1','SP'), (2,'sp2','SP'), (3,'sp3','SP'), (4,'tr1','TR'),
(5,'tr2','TR'), (6,'tbl1','U')

-- Running a query
SELECT   FIRST_VALUE(ObjectName) OVER(PARTITION BY ObjectType ORDER BY
ObjectType ) as FirstValue,
ObjectType, ObjectName
FROM TESTCTP3
GO
```

1.6.3 LAG

LAG () accesses data from a previous row in the same result set without the use of a self-join. LAG provides access to a row at a given physical offset that comes before the current row. Use this analytic function in a SELECT statement to compare values in the current row with values in a previous row.

Syntax

LAG (scalar_expression [,offset] [,default]) OVER ([partition_by_clause] order_by_clause)

- Scalar_expression is the value to be returned based on the specified offset.
- Offset is the number of rows back from the current row from which to obtain a value. If not specified, the default is 1.
- Default is the value to return when scalar_expression at offset is NULL.
- Partition_by_clause divides the result set produced by the FROM clause into partitions to which the function is applied.
- Order_by_clause determines the order of the data before the function is applied.

Sample code

```
CREATE TABLE #Orders (
   OrderDate     DATE,
   ProductID     INT,
   Quantity      INT);

INSERT INTO #Orders VALUES
```

```
('2011-07-28',11,12), ('2011-03-18',12,74), ('2011-04-12',13,95),
('2011-07-25',14,57), ('2011-05-30',11,28), ('2011-05-21',10,12),
('2011-04-12',11,38);

SELECT OrderDate, ProductID, Quantity, LAG(OrderDate, 1, OrderDate)
       OVER (PARTITION BY ProductID ORDER BY OrderDate)
       AS Last_OrderDate
FROM   #Orders
```

1.6.4 LAST_VALUE

LAST_VALUE returns the last value in an ordered set of values.

Syntax

LAST_VALUE ([scalar_expression) OVER ([partition_by_clause] order_by_clause rows_range_clause)

- Scalar_expression is the value to be returned.
- Partition_by_clause divides the result set produced by the FROM clause into partitions to which the function is applied.
- Order_by_clause determines the order of the data before the function is applied.
- Rows_range_clause further limits the rows within the partition by specifying start and end points.

Sample code

```
CREATE TABLE TESTCTP3 (
Objectid int,
ObjectName varchar(500),
ObjectType varchar(100)) ;

-- Inserting sample records
INSERT INTO testctp3
VALUES (1,'sp1','SP'), (2,'sp2','SP'), (3,'sp3','SP'), (4,'tr1','TR'),
(5,'tr2','TR'), (6,'tbl1','U');

SELECT
LAST_VALUE(ObjectName) over(partition by ObjectType order by ObjectType ) as
LastValue,
ObjectType ,ObjectName
FROM TESTCTP3
```

1.6.5 LEAD

LEAD accesses data from a subsequent row in the same result set without the use of a self-join. Use this analytic function in a SELECT statement to compare values in the current row with values in a following row.

Syntax

LEAD (scalar_expression [,offset] , [default]) OVER ([partition_by_clause])

- Scalar_expression is the value to be returned based on the specified offset.
- Offset is the number of rows forward from the current row from which to obtain a value.
- Default is the value to return when scalar_expression at offset is NULL.
- Partition_by_clause divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group.
- Order_by_clause determines the order of the data before the function is applied.

Sample code

```
CREATE TABLE #Orders (
   OrderDate     DATE,
   ProductID     INT,
   Quantity      INT);

INSERT INTO #Orders VALUES
('2011-07-28', 11, 12), ('2011-03-18', 12, 74), ('2011-04-12', 13, 95),
('2011-07-25', 14, 57), ('2011-05-30', 11, 28), ('2011-05-21', 10, 12),
('2011-04-12', 11, 38);

SELECT OrderDate, ProductID, Quantity, LEAD (OrderDate, 1, OrderDate)
OVER (PARTITION BY ProductID ORDER BY OrderDate) AS Next_OrderDate
FROM   #Orders;
```

1.6.6 PERCENTILE_CONT

PERCENTILE_CONT calculates a percentile based on a continuous distribution of the column. The result is interpolated and might not be equal to any of the specific values in the column.

Syntax

PERCENTILE_CONT (numeric_literal)

WITHIN GROUP (ORDER BY order_by_expression [ASC | DESC])

OVER ([<partition_by_clause>])

- Numeric_literal is the percentile to compute. The value must range between 0.0 and 1.0.
- WITHIN GROUP **(** ORDER BY order_by_expression [ASC | DESC]**)** Specifies a list of numeric values to sort and compute the percentile over
- OVER **(** <partition_by_clause> **)** Divides the result set produced by the FROM clause into partitions to which the percentile function is applied.

Sample code

```
CREATE TABLE [Results] (
[Subject] varchar(10),
[Student] varchar(50),
[Marks] int )

INSERT INTO [Results]
VALUES ('Maths','Student1',45), ('Physics','Student2',45),
('Physics','Student1',50), ('Chemistry','Student3',20),
('Physics','Student3',35), ('Biology','Student1',20),
('Biology','Student2',60), ('Biology','Student3',65),
('Chemistry','Student1',75), ('Biology','Student4',30)
GO

SELECT  [Student], [Subject], [Marks],
PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY [MARKS])
OVER(PARTITION BY [Subject])
FROM [Results]
ORDER BY [Subject], [Marks]
GO
```

1.6.7 PERCENTILE_DISC

PERCENTILE_DISC computes a specific percentile for sorted values in an entire row-set or within distinct partitions of a row-set.

Syntax

PERCENTILE_DISC (numeric_literal) WITHIN GROUP (ORDER BY order_by_expression [ASC | DESC])
OVER ([<partition_by_clause>])

- Literal is the percentile to compute. The value must range between 0.0 and 1.0.
- WITHIN GROUP **(**ORDER BY order_by_expression [ASC | DESC]**)** Specifies a list of numeric values to sort and compute the percentile over.
- OVER **(**<partition_by_clause>) divides the result set produced by the FROM clause into partitions to which the percentile function is applied.

Sample code

```
SELECT  [Student], [Subject], [Marks],
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY [MARKS])
OVER(PARTITION BY [Subject]) as PD
FROM [Results]
ORDER BY [Subject], [Marks]
```

1.6.8 PERCENT_RANK

PERCENT_RANK calculates the relative rank of a row within a group of rows. Use PERCENT_RANK to evaluate the relative standing of a value within a query result set or partition.

Syntax

PERCENT_RANK( ) OVER ([partition_by_clause] order_by_clause)

- Partition_by_clause divides the result set produced by the FROM clause into partitions to which the function is applied.
- Order_by_clause determines the logical order in which the operation is performed.

Sample code

```
SELECT [Student],[Subject],[Marks], PERCENT_RANK() OVER(PARTITION BY
[Subject] ORDER BY [Marks]) as PR
FROM [Results]
ORDER BY [Subject],[Marks]
GO
```

2 SEQUENCE

2.1 Introduction

To generate a sequence of numeric values, until SQL Server 2008 R2, IDENTITY property defined on a column was used. While this has its own advantages, there are a few disadvantages –

1. Sharing the newly generated sequence values in a table with other tables required usage of global variables, thus making it difficult.

2. Migrating from other databases like Oracle or DB2 poses a challenge with respect to a database object called "SEQUENCE".

To overcome these disadvantages, SQL Server 2011 is being endowed with a new feature called SEQUENCE, which is similar to other databases in certain ways and has its own distinct properties as well.

A sequence is a user-defined schema-bound object that generates a sequence of numeric values according to the specification with which the sequence was created. The sequence of numeric values is generated in an ascending or descending order at a defined interval and may cycle (repeat) as requested. Generally it is used to create auto generated Primary key (surrogate key) value.

2.2 Syntax

2.2.1 CREATE SEQUENCE

As usual, you have TSQL command and GUI. TSQL Syntax is as follows

```
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 } ]
   [ ; ]
```

This MSDN link has more information on the syntax -
http://msdn.microsoft.com/en-us/library/ff878091(v=SQL.110).aspx

Here are a few examples -

Creating a Sequence with all arguments

```
CREATE SEQUENCE [dbo].Seq_EmployeeID
```

```
        START WITH 0
        INCREMENT BY 2
        MINVALUE 0
        MAXVALUE 2000
        CYCLE
        CACHE  20

Creating Table with Sequence as Default value for PK

CREATE TABLE Audit.ProcessEvents
(
    EventID int PRIMARY KEY CLUSTERED
        DEFAULT (NEXT VALUE FOR Audit.EventCounter),
    EventTime datetime NOT NULL DEFAULT (getdate()),
    EventCode nvarchar(5) NOT NULL,
    Description nvarchar(300) NULL
) ;
GO
```

2.2.2 ALTER SEQUENCE

Modifies the arguments of an existing sequence object. If the sequence was created with the **CACHE** option, altering the sequence will recreate the cache. http://msdn.microsoft.com/en-us/library/ff878572(v=sql.110).aspx

2.2.3 DROP SEQUENCE

After generating a number, a sequence object has no continuing relationship to the number it generated, so the sequence object can be dropped, even though the number generated is still in use. http://msdn.microsoft.com/en-us/library/ff878471(v=sql.110).aspx

2.2.4 Other system Functions and Objects related to Sequence

2.2.4.1 NEXT VALUE FOR

This command generates the next sequence number from the specified sequence object. The NEXT VALUE FOR function can be used in stored procedures and triggers. This function also have few limitations as explained in http://msdn.microsoft.com/en-us/library/ff878370(v=sql.110).aspx

2.2.4.2 sys.sequence

This system table contains a row for each sequence object in a database.
Ref: http://msdn.microsoft.com/en-us/library/ff877934(v=sql.110).aspx

2.2.4.3 sp_sequence_get_range

This system stored procedure returns a range of sequence values from a sequence object. The sequence object generates and issues the number of values requested and provides the application with metadata related to the range.

2.3 Important Architectural Consideration and Limitations

a) By default, SEQUENCE uses BIGINT data type which consumes 8 bytes. If the requirement is to store smaller values, then INT, SMALLINT or TINYINT data type needs to be used accordingly.
b) If the starting value is not defined while creating, then SQL Server set it to the lowest value that the data type can support. So, if the data type is BIGINT the starting number will be (-9,223,372,036,854,775,808)
c) The Increment cannot be 0. If the increment is a negative value, the sequence object is descending; otherwise, it is ascending. By default the increment is 1
d) If the requirement is to migrate Oracle Sequence to SQL Server, then select the data type as Numeric(28,0)
e) Specify CYCLE/NOCYCLE to define the behavior if the sequence number reaches the maximum value
f) SQL Server as of CTP 3 does not support Current Value
g) Cache option is different from Oracle. Oracle, by default, caches 20 Sequence objects but in SQL Server database engine determines the size. If the cache option is enabled without specifying a cache size, the Database Engine will select a size. However, users should not rely upon the selection being consistent. Microsoft might change the method of calculating the cache size without notice.
h) Sequence objects support ownership chaining. If the sequence object has the same owner as the calling stored procedure, trigger, or table (having a sequence object as a default constraint), no permission check is required on the sequence object. If the sequence object is not owned by the same user as the calling stored procedure, trigger, or table, a permission check is required on the sequence object
i) Altering a column and Adding Identity is not possible in SQL Server. In such a scenario, a new column with IDENTITY property needs to be created and then updated with values from the existing column where the property was desired; then, the old column should be dropped and the new one renamed. When such an option is not possible, then may be adding SEQUENCE as a default can be a workaround

2.4 How does it differ from Identity Column?

Sequences, unlike identity columns, are not associated with tables. An application refers to a sequence object to receive its next value. The relationship between sequences and tables is controlled by the application. User applications can reference a sequence object and coordinate the values keys across multiple rows and tables.

Unlike identity columns, whose values cannot be changed, sequence values are not automatically protected after insertion into the table. To prevent sequence values from being changed, use an update trigger on the table to roll back changes.

2.5 Performance Comparison of Sequence and Identity

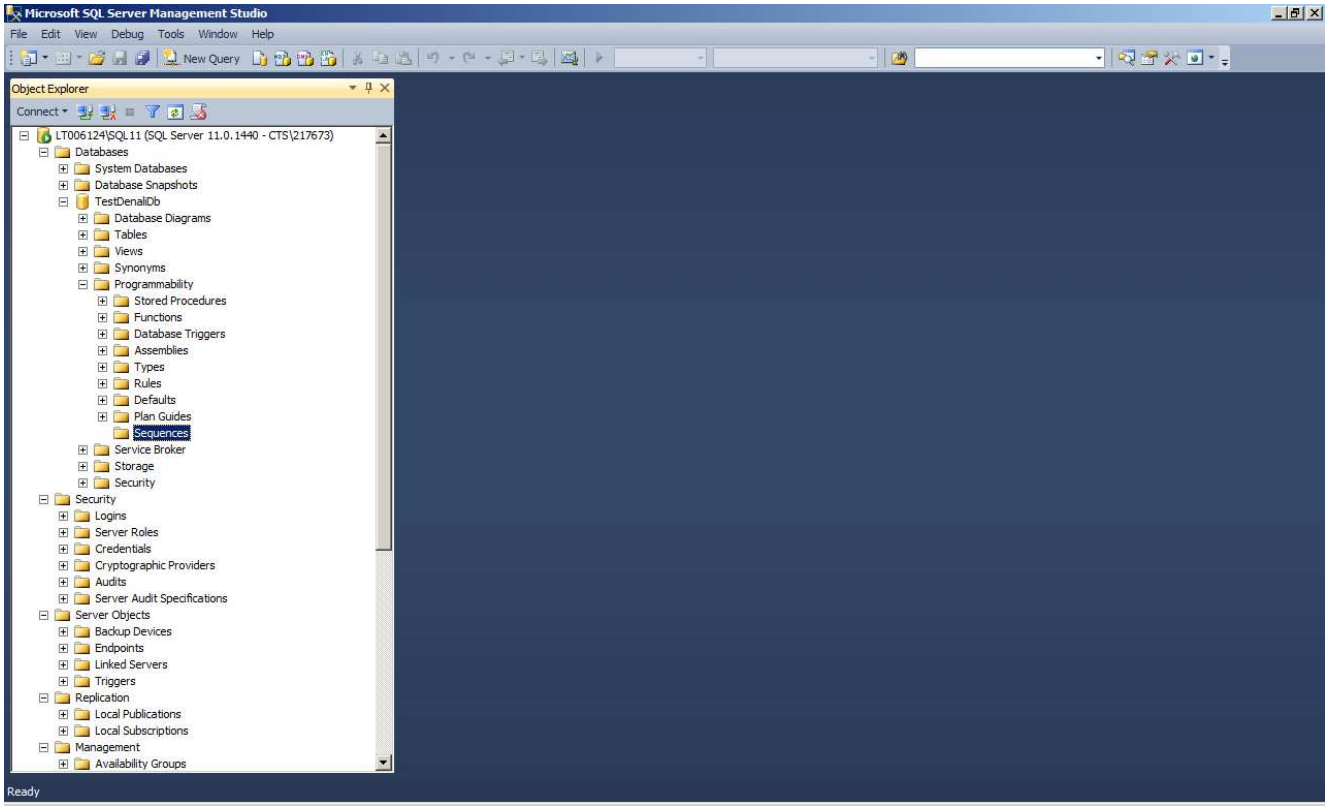
Test carried out – A table with IDENTITY column has been compared with a table with SEQUENCE set as default for PK (no default was provided). In this case, insertion into the former table with IDENTITY was 60% faster than the latter. Script for the test is attached below –

```
-- Table with sequence as default
CREATE TABLE [dbo].[SequencePerf](
      [EventID] [int] NOT NULL,
      [Description] [nvarchar](300) NULL,
PRIMARY KEY CLUSTERED ( [EventID] ASC)
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[SequencePerf] ADD DEFAULT (NEXT VALUE FOR [Seq_EventId])
FOR [EventID]
GO
CREATE TABLE [dbo].[IdentityPerf](
      [EventID] [int] IDENTITY(1,1) NOT NULL,
      [Description] [nvarchar](300) NULL,
PRIMARY KEY CLUSTERED ( [EventID] ASC)
) ON [PRIMARY]
GO
-- Run both the insert in single batch and see the cost it took. In our case
it was 71:29
Insert [SequencePerf] (Description)
Select top 100000 o.name From sys.objects o,sys.columns

Insert [IdentityPerf] (Description)
Select top 100000 o.name From sys.objects o,sys.columns
```

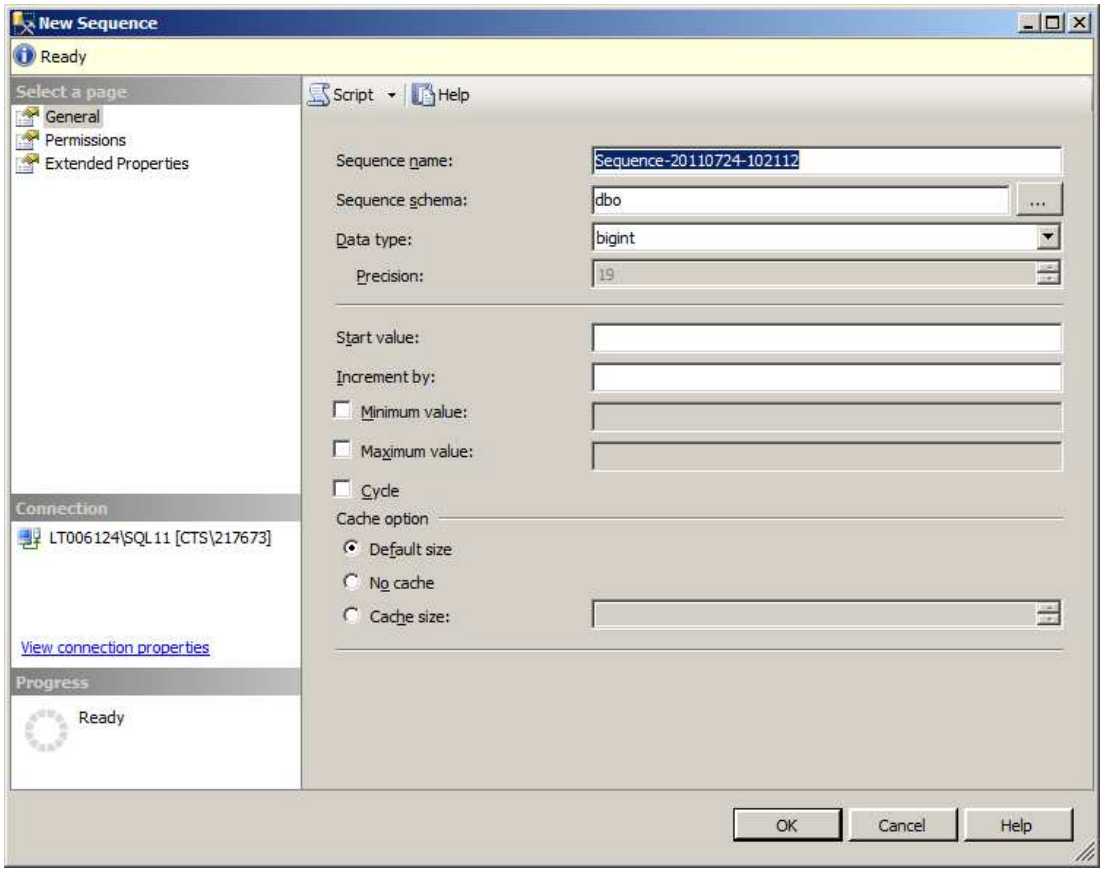
Aaron Bertrand has covered this in great detail in his blog here - http://sqlblog.com/blogs/aaron_bertrand/archive/2010/11/11/sql-server-11-denali-using-sequence.aspx

2.6 How to create Sequence from Management Studio

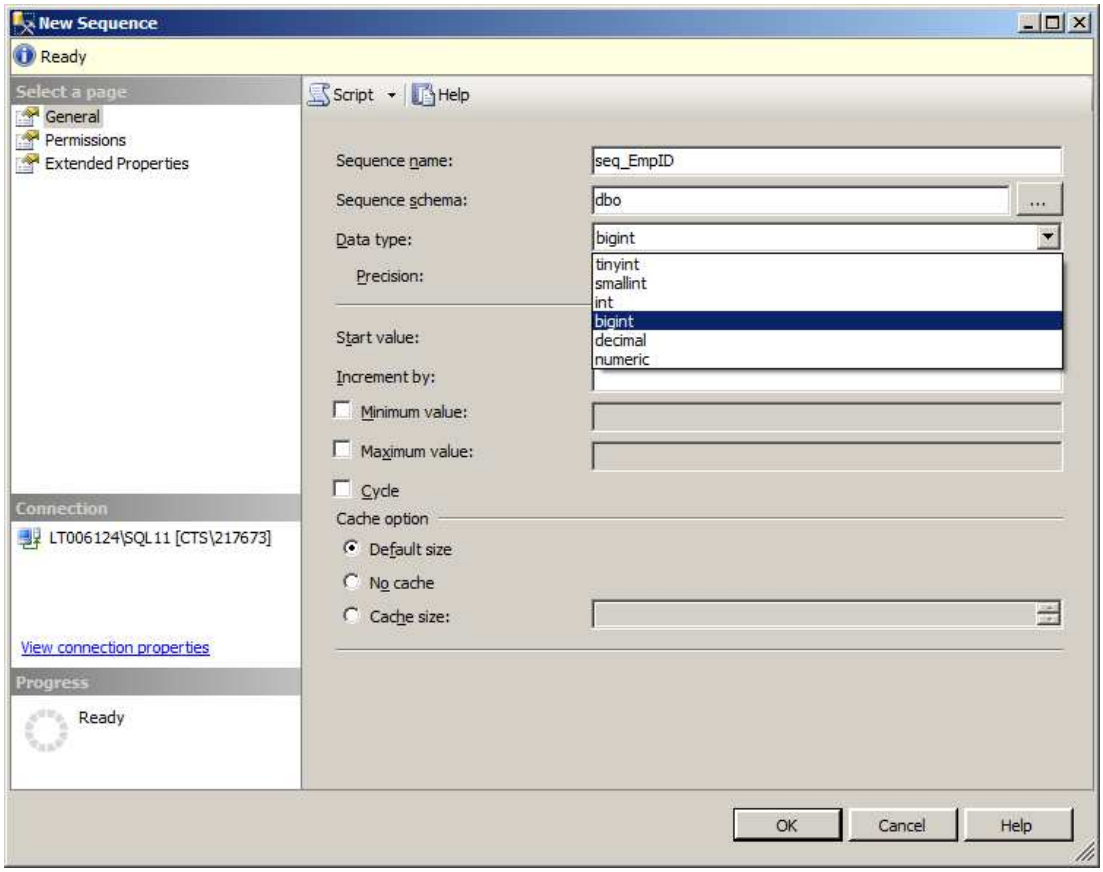
2.6.1 Sequences comes under "Programmability"



2.6.2 New Sequence window



2.6.3 Data type to be chosen carefully



2.7 Different Scenario of Sequence Usage in SQL Server

```
Create database [TestDenali]
GO

USE [TestDenali]
GO
              CREATE SEQUENCE [dbo].[Seq_EventId]
               AS [bigint]
               START WITH 1
               INCREMENT BY 1
               MINVALUE 1
               MAXVALUE 9223372036854775807
               CYCLE
               CACHE  200
              GO
              CREATE TABLE [dbo].[SequenceTest](
                     [EventID] [int] NOT NULL ,
                     [Description] [nvarchar](300) NULL,
              PRIMARY KEY CLUSTERED
              (
                     [EventID] ASC
              )
              ) ON [PRIMARY]
              GO
              ALTER TABLE [dbo].[SequenceTest] ADD  DEFAULT (NEXT VALUE FOR
[Seq_EventId]) FOR [EventID]
              GO
-- Not providing EventID value so that Default
              INSERT INTO dbo.[SequenceTest](Description)
               select top 100 Name from sys.objects
--  Inserting bulk number of records , getting sequence in Select
              INSERT INTO dbo.[SequenceTest] ([EventID],description)
              SELECT   top 100   NEXT VALUE FOR  Seq_EventId AS SecondUse, Name
              from sys.objects

-- Insert a single row

              INSERT  [SequenceTest] ([EventID],description)
```

```
                    VALUES (NEXT VALUE FOR Seq_EventId, 'Test') ;

-- Usage with SELECT INTO
              SELECT NEXT VALUE FOR Seq_EventId AS EventID, Description
                     INTO [SequenceTest_clone]
                     FROM [SequenceTest] ;
-- Temp table
              SELECT NEXT VALUE FOR Seq_EventId AS EventID, Description
                     INTO [#SequenceTest_clone]
                     FROM [SequenceTest] ;

-- Creating Temp table with Sequence as Defualt
              CREATE TABLE #SequenceTest  (
                     [EventID] [int] NOT NULL DEFAULT (NEXT VALUE FOR Seq_EventId) ,
                     [Description] [nvarchar](300) NULL,
              PRIMARY KEY CLUSTERED
              (
                     [EventID] ASC
              )
              ) ON [PRIMARY]

              INSERT  #SequenceTest ([EventID],description)
                     VALUES (NEXT VALUE FOR Seq_EventId, 'Test') ;
              INSERT  #SequenceTest ( description)
                     VALUES (   'Test') ;
              Select *from #SequenceTest

-- Creating Table Variable with Sequence as Default
              Declare  @SequenceTest TABLE (
                     [EventID] [int] NOT NULL DEFAULT (NEXT VALUE FOR Seq_EventId) ,
                     [Description] [nvarchar](300) NULL)

              INSERT  @SequenceTest ([EventID],description)
              VALUES (NEXT VALUE FOR Seq_EventId, 'Test') ;
              INSERT  @SequenceTest ( description)
              VALUES (   'Test') ;

              Select *From @SequenceTest
```

```
-- Table Valued Parameter
CREATE TYPE [dbo].[SequenceTestTableType] AS TABLE
(
    [EventID] INT DEFAULT (NEXT VALUE FOR Seq_EventId), description VARCHAR(128)
)
/*
Msg 11719, Level 15, State 1, Line 3
NEXT VALUE FOR function is not allowed in check constraints, default objects, computed columns, views,
user-defined functions, user-defined aggregates, sub-queries, common table expressions, or derived tables.
*/

Create view vwTestSequence
as
SELECT   top 10000  NEXT VALUE FOR  Seq_EventId AS SecondUse,description FROM dbo.Sequencetest

/*
Msg 11719, Level 15, State 1, Procedure vwTestSequence, Line 4
NEXT VALUE FOR function is not allowed in check constraints, default objects, computed columns, views,
 user-defined functions, user-defined aggregates, sub-queries, common table expressions, or derived tables.

*/
-- Capture the Inserted Sequencenumber using OUTPUT. In SuperType-Subtype scenario may be useful
Declare @tab table (Eventid int)
Insert [SequencePerf] (Description) OUTPUT inserted.[EventID] Into @tab
Select top 100000 o.name From sys.objects o,sys.columns
SELECT *From @tab
```

3 Exception handling enhancement in SQL Server 2011

3.1 THROW statement

THROW() raises an exception and transfers execution to a CATCH block of a TRY…CATCH construct in Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3).

Syntax

THROW [{ error_number | @local_variable },

{ message | @local_variable },

{ state | @local_variable }

] [;]

- Error_number is a constant or variable that represents the exception. error_number is int and must be greater than or equal to 50000 and less than or equal to 2147483647.
- Message is an string or variable that describes the exception. message is nvarchar(2048).
- State is a constant or variable between 0 and 255 that indicates the state to associate with the message. State is tinyint.

Sample code

Using THROW to raise an exception

```
The following example shows how to use the THROW statement to raise an
exception.

THROW 51000, 'The record does not exist.', 1;
```

Using THROW to raise an exception again

```
The following example shows how use the THROW statement to raise the last
thrown exception again.

USE tempdb;
GO
CREATE TABLE dbo.TestRethrow
(    ID INT PRIMARY KEY
);
BEGIN TRY
    INSERT dbo.TestRethrow(ID) VALUES(1);
```

```
--  Force error 2627, Violation of PRIMARY KEY constraint to be raised.
    INSERT dbo.TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH

    PRINT 'In catch block.';
    THROW;
END CATCH;

PRINT 'In catch block.';

Msg 2627, Level 14, State 1, Line 1

Violation of PRIMARY KEY constraint 'PK__TestReth__3214EC272E3BD7D3'. Cannot
insert duplicate key in object 'dbo.TestRethrow'.

The statement has been terminated.
```

Using FORMATMESSAGE with THROW

```
The following example shows how to use the FORMATMESSAGE statement with THROW
to throw a customized error message.

DECLARE @Message NVARCHAR(2048);
SELECT @Message = FORMATMESSAGE(1127);
THROW 50001, @Message, 1;
```

3.2 Comparisons between Try…Catch model and Throw

In previous releases SQL Server 2005, 2008 and 2008 R2, we used TRY/CATCH for Exception Handling in TSQL statements.

The drawbacks of Raiserror have been overcome by the new Throw command of Denali. One of the drawbacks of the Raiserror that we encountered was that it does not retain the original error line number. Let us observe how far the fact is true while using Throw command.

Let us see an example of TRY/CATCH in a scenario where we raise an exception when a NULL value is entered into [Phone Number] column of temporary table #tblExceptionTest.

```
--If the #tblExceptionTest object exists in the tempdb, then drop it
If OBJECT_ID ('tempdb..#tblExceptionTest') Is not null
Begin
      Drop Table #tblExceptionTest
End

Begin TRY
   --Create the #tblExceptionTest temporary table
      Create Table #tblExceptionTest (Id int identity, [Phone Number]
varchar(10) not null)

    Begin Transaction TranExcpHandlingTest_2005_2008
         --Variable Declarations
       Declare @i int -- a local variable that acts as a counter
         --Initialize variables
       Set @i =1
       --Start Operation
       While (@i <= 4)
        Begin
                  -- Simulating the situation where a user tries to enter a
null value to the Phone Number column
            If (@i = 4)
                Begin
                    Insert into #tblExceptionTest ([Phone Number]) Values
(null)
                End
            Else
                -- All records will be inserted successfully
                Begin
```

```
                    Insert into #tblExceptionTest ([Phone Number]) Values
(cast (@i as varchar (2)) + '12345678')
                End
            Set @i = @i +1
        End -- End of while

            --If everything goes smooth, then commit the transaction
        Commit Transaction TranExcpHandlingTest_2005_2008
End Try
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
              --Raise the custom error
          RAISERROR ('Attempt to insert null value in [Phone Number] is not
allowed', 16,1)
       End
End Catch

--Display the records
Select * From #tblExceptionTest

After executing the above code, we get the following result

(1 row(s) affected)
(1 row(s) affected)
(1 row(s) affected)
(0 row(s) affected)
Msg 50000, Level 16, State 1, Line 45
Attempt to insert null value in [Phone Number] is not allowed

(0 row(s) affected)
```

We create our own custom message that is defined in the Raiserror function. The valid action part of the logic goes into the TRY block and the error handling part resides into the Catch block. If the piece of code within the Try block seems to be anomalous, the control goes to the Catch block, Rolls back the transaction and the rest of the program resumes. If all the statement within the Try block runs smoothly, then the control never enters the Catch block but executes the very first statement following

the End Catch statement. Moreover, the catch block provides sufficient information during the anomalous situation which should be trapped for appropriate information to glean about the program failure like

1. ERROR_NUMBER
2. ERROR_SEVERITY
3. ERROR_STATE
4. ERROR_LINE
5. ERROR_PROCEDURE
6. ERROR_MESSAGE

If we change the CATCH block like below in the above program,

```
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
              SELECT
                        ERROR_NUMBER() AS ErrorNumber,
                        ERROR_SEVERITY() AS ErrorSeverity,
                        ERROR_STATE() AS ErrorState,
                        ERROR_PROCEDURE() AS ErrorProcedure,
                        ERROR_LINE() AS ErrorLine,
                        ERROR_MESSAGE() AS ErrorMessage;
       End
End Catch
```

We get the following result:

	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	515	16	2	NULL	24	Cannot insert the value NULL into column 'Phone ...

As an alternative proof of the concept, let us change the CATCH block as below

```
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
         DECLARE @errNumber INT = ERROR_NUMBER()
         DECLARE @errMessage VARCHAR(500) = 'Attempt to insert null value in
[Phone Number] is not allowed'
         --Raise the custom error
        RAISERROR('Error Number: %d, Message: %s', 16, 1, @errNumber,
@errMessage)
       End
End Catch
```

We get the following result

```
(1 row(s) affected)
(1 row(s) affected)
(1 row(s) affected)
(0 row(s) affected)
Msg 50000, Level 16, State 1, Line 47
Error Number: 515, Message: Attempt to insert null value in [Phone Number] is
not allowed
```

So we can conclude that, by using Raiserror, we lose the original error line number.

Rewriting the Catch block of the above T-Sql code using Throw –

```
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2011;
              --Throw the error
          THROW
       End
End Catch

We get the below result

Msg 515, Level 16, State 2, Line 24
Cannot insert the value NULL into column 'Phone Number', table
'tempdb.dbo.#tblExceptionTest_____________________________________
_____________________000000000018'; column does not allow nulls. INSERT
fails.
```

This is rather correct and hence proves our statement.

Also, using Raiserror we cannot re-raise the same error because RAISE ERROR expects the number to be stored in the sys.messages. Throw does not expect the error number to be in the sys.messages table though the error number should be between 50000 and 2147483647(both inclusive). THROW inside a CATCH block acts like RETHROW - it will re-raise the exception that transferred to the CATCH block in the first place.

```
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
              --Raise the custom error
              RAISERROR(515, 16, 1)
       End
End Catch
```

```
We receive the below error message -

(0 row(s) affected)
Msg 2732, Level 16, State 1, Line 45
Error number 515 is invalid. The number must be from 13000 through 2147483647
and it cannot be 50000.
```

As a part of this exercise, let's change the Catch block as under

```
Begin Catch
        --Handle the error
       Begin
              --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2011;
              --Throw the error
          THROW 50001,'Attempt to insert null value in [Phone Number] is not
allowed',1
       End
End Catch

We get the result as
Msg 50001, Level 16, State 1, Line 45
Attempt to insert null value in [Phone Number] is not allowed
```

Though there are now many ways to handle exception is Sql Server, still every error that arises is not being caught by the Try...Catch construct. For example

a) Syntax error is being caught by the Query Editor parser of SSMS
b) Invalid object names

For example, if we do something as

```
Begin Try
       -- --Invalid object tblInvalid
     Insert Into tblInvalid(Id,DOB) Values(1,DATEADD(year,1,'2011-02-26'))
End Try
Begin Catch
       --Throw the error
     THROW
End Catch
```

```
And try to execute the same, we will receive the below error

Msg 208, Level 16, State 1, Line 3
Invalid object name 'tblInvalid'.
```

So we can make out that, it is nearly impossible to trap such kind of errors.

But there is a tricky way of doing so. The idea is to create two stored procedures, call one inside the other in the Try...Catch block and trap the exception. As a proof of the above statement, we will take the above scenario into consideration and will go ahead to do the experiment.

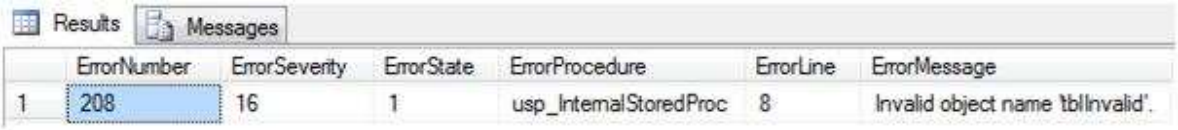
```
--Check if the stored procedure exists. If so drop it
If Exists (Select * from sys.objects where name = 'usp_InternalStoredProc'
and type = 'P')
    Drop Procedure usp_InternalStoredProc
Go
-- Create the internal stored procedure
Create Procedure usp_InternalStoredProc
As
Begin
    Begin Transaction TranExcpHandlingTest_2011
       Begin Try
                  --Invalid object tblInvalid
                  Insert Into tblInvalid(Id,DOB)
Values(1,DATEADD(year,1,'2011-02-26'))
                  --Commits the transaction
                  Commit Transaction TranExcpHandlingTest_2011
            End Try
       Begin Catch
              If @@TRANCOUNT > 0 Rollback Transaction
TranExcpHandlingTest_2011
          Print 'In catch block of internal stored procedure.... throwing the
exception';
          -- Throw the exception
          THROW
       End Catch
End
Go

-- Script for creating the External stored procedure
```

```
--Check if the stored procedure exists. If so drop it
If Exists (Select * from sys.objects where name = 'usp_ExternalStoredProc'
and type = 'P')
    Drop Procedure usp_ExternalStoredProc
Go
-- Create the external stored procedure
Create Procedure usp_ExternalStoredProc
As
Begin
    Begin Try
       --Call the internal stored procedure
        Exec usp_InternalStoredProc
    End Try
    Begin Catch
        Print 'In catch block of external stored procedure.... throwing the
exception';
            SELECT ERROR_NUMBER() AS ErrorNumber
                ,ERROR_SEVERITY() AS ErrorSeverity
                ,ERROR_STATE() AS ErrorState
                ,ERROR_PROCEDURE() AS ErrorProcedure
                ,ERROR_LINE() AS ErrorLine
                ,ERROR_MESSAGE() AS ErrorMessage;
        THROW
    End Catch
End
Go
--Executing the outer procedure
Exec usp_ExternalStoredProc
```

The output is as below

```
In catch block of external stored procedure.... throwing the exception

(1 row(s) affected)
Msg 208, Level 16, State 1, Procedure usp_InternalStoredProc, Line 8
Invalid object name 'tblInvalid'.
```

The Result pane gives the below



Results | Messages

	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	208	16	1	usp_InternalStoredProc	8	Invalid object name 'tblInvalid'.

Code explanation

We have two stored procedures, usp_InternalStoredProc and usp_ExternalStoredProc. In the usp_InternalStoredProcedure, we are trying to insert record into table (#tblInnerTempTable) which does not have any existence. From the usp_InternalStoredProcedure, we are calling the inner procedure and the exception is being caught in the outer procedures catch block.

Moreover, the error line (which is 8 here) is also correct.

Learn more about differences between throw and try…catch model from below link

http://sqlblog.com/blogs/aaron_bertrand/archive/2010/11/22/sql-server-v-next-denali-using-throw-instead-of-raiserror.aspx

3.3 XACT_ABORT IN THROW

As every T-SQL programmer should know, an exception does not roll back a transaction by default (though it does depend on severity level to an extent – but a "normal" exception does not roll back a transaction). I.e. the following code would cause two rows to be inserted in the table t1:

```
--first create a test table which we will use throughout the code samples
CREATE TABLE t1 (id int primary key, col1 nvarchar(15));
--now onto the 'meat'
BEGIN TRAN
INSERT INTO t1 VALUES(1, 'row1');
--emulate some error, this will indeed cause an exception to happen,
--but the processing will continue
SELECT 1 / 0
INSERT INTO t1 VALUES(2, 'row2')
COMMIT

The output is as below
 (1 row(s) affected)
Msg 8134, Level 16, State 1, Line 8
```

```
Divide by zero error encountered.

(1 row(s) affected)


SELECT * FROM t1;
```

	id	col1
1	1	row1
2	2	row2

We can indicate that we want "automatic" rollback of transactions when an exception happens by setting XACT_ABORT. This will cause a rollback to happen if a system exception happens. So based on the example above, no rows will be inserted when the code below executes:

```
SET XACT_ABORT ON
BEGIN TRAN
INSERT INTO t1 VALUES(3, 'row3');
SELECT 1 / 0
INSERT INTO T1 VALUES(4, 'row4')
COMMIT

The output is as below
 (1 row(s) affected)
Msg 8134, Level 16, State 1, Line 4
Divide by zero error encountered.

SELECT * FROM t1;
```

	id	col1
1	1	row1
2	2	row2

However, what happens if the user throws an exception using RAISERROR?! In that case no rollback happens, i.e. RAISERROR does not honor the XACT_ABORT setting:

```
SET XACT_ABORT ON
BEGIN TRAN
INSERT INTO t1 VALUES(5, 'row5');
--the user raises an error, but the tx will not roll back
RAISERROR('Oooops', 16, 1)
INSERT INTO t1 VALUES(6, 'row6')
COMMIT

The output is as below
(1 row(s) affected)
Msg 50000, Level 16, State 1, Line 5
Oooops
(1 row(s) affected)

SELECT * FROM t1;
```

	id	col1
1	1	row1
2	2	row2
3	5	row5
4	6	row6

This is a fairly severe drawback.

So with the introduction of Denali / SQL 11 and the THROW keyword, **Microsoft has tried to fix this by making THROW honor XACT_ABORT:**

```
SET XACT_ABORT ON
BEGIN TRAN
INSERT INTO t1 VALUES(7, 'row7');
--the user raises an error, and the tx will roll back
THROW 50000, 'Ooops', 1
INSERT INTO t1 VALUES(8, 'row8')
COMMIT

The output is as below
```

```
(1 row(s) affected)
Msg 50000, Level 16, State 1, Line 5
Ooops

SELECT * FROM t1;
```

	id	col1
1	1	row1
2	2	row2
3	5	row5
4	6	row6

3.4 Points to remember

- We need to terminate the previous batch by placing a semi colon before THROW as THROW command should be issued as a new batch; else we get an error.
- Some functionality on Raiserror is now in future deprecation list.
- FORMATMESSAGE () ensures that there is no duplicity of error messages in sys.messages.
- Do not define a severity when using THROW, all exceptions being raised by THROW has a severity of 16.
- Using throw we can throw a specific error number as well as message.
- When using throw we have to define both an error number as well as a message (state) unless we re-throw an exception.
- The error number does not have to exist in sys.messages but it has to be between 50000 and 2147483647.
- THROW inside a CATCH block acts like RETHROW - it will re-raise the exception that transferred to the CATCH block in the first place.
- If a TRY…CATCH construct is not available, the session is ended. The line number and procedure where the exception is raised are set.
- If the THROW statement is specified without parameters, it must appear inside a CATCH block. This causes the caught exception to be raised. Any error that occurs in a THROW statement causes the statement batch to be ended.

4 Change Startup Parameter

Startup options designate certain file locations needed during startup, and specify some server wide conditions. Most users do not need to specify startup options unless you are troubleshooting the Database Engine or you have an unusual problem and are directed to use a startup option by SQL Server Customer Support

Changing SQL Services Startup Parameters is very easy in SQL Server Denali, there is no need to take care of syntax; This will be taken care automatically by SQL Server.

When you install SQL Server, Setup writes a set of default startup options in the Microsoft Windows registry. You can use these startup options to specify an alternate master database file, master database log file, or error log file. If the Database Engine cannot locate the necessary files, SQL Server will not start.

Startup options can be set by using SQL Server Configuration Manager.

4.1 List of Startup Options

i. -d master_file_path

Fully qualified path for the master database file (typically, C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Data\master.mdf). If you do not provide this option, the existing registry parameters are used.

ii. -e error_log_path

Fully qualified path for the error log file (typically, C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\LOG\ERRORLOG). If you do not provide this option, the existing registry parameters are used.

iii. -l master_log_path

Fully qualified path for the master database log file (typically C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Data\mastlog.ldf). If you do not specify this option, the existing registry parameters are used.

iv. -c

Shortens startup time when starting SQL Server from the command prompt.

v. -f

Starts an instance of SQL Server with minimal configuration. This is useful if the setting of a configuration value (for example, over-committing memory) has prevented the server from starting. Starting SQL Server in minimal configuration mode places SQL Server in single-user mode.

vi. -g memory_to_reserve

Specifies an integer number of megabytes (MB) of memory that SQL Server will leave available for memory allocations within the SQL Server process, but outside the SQL Server memory pool. Use of this option might help tune memory allocation, but only when physical memory exceeds the configured limit set by the operating system on virtual memory available to applications.

vii. -m

Starts an instance of SQL Server in single-user mode. When you start an instance of SQL Server in single-user mode, only a single user can connect, and the CHECKPOINT process is not started. CHECKPOINT guarantees that completed transactions are regularly written from the disk cache to the database device.

viii. -m"Client Application Name"

Limits the connections to a specified client application, when you use the -m option with SQLCMD or SQL Server Management Studio. Client Application Name is case sensitive.Do not use this option as a security feature. The client application provides the client application name, and can provide a false name as part of the connection string.

ix. -n

Does not use the Windows application log to record SQL Server events. If an instance of SQL Server is started with -n, it is recommended to also use the -e startup option. Otherwise, SQL Server events are not logged.

x. -s

Allows you to start a named instance of SQL Server. Without the -s parameter set, the default instance will try to start. You must switch to the appropriate BINN directory for the instance at a command prompt before starting sqlservr.exe

xi. -T trace#

Indicates that an instance of SQL Server should be started with a specified trace flag (trace#) in effect. Trace flags are used to start the server with nonstandard behavior.

xii. -x

Disables the following monitoring features:

- SQL Server performance monitor counters
- Keeping CPU time and cache-hit ratio statistics
- Collecting information for the DBCC SQLPERF command
- Collecting information for some dynamic management views
- Many extended-events event points

xiii. -E

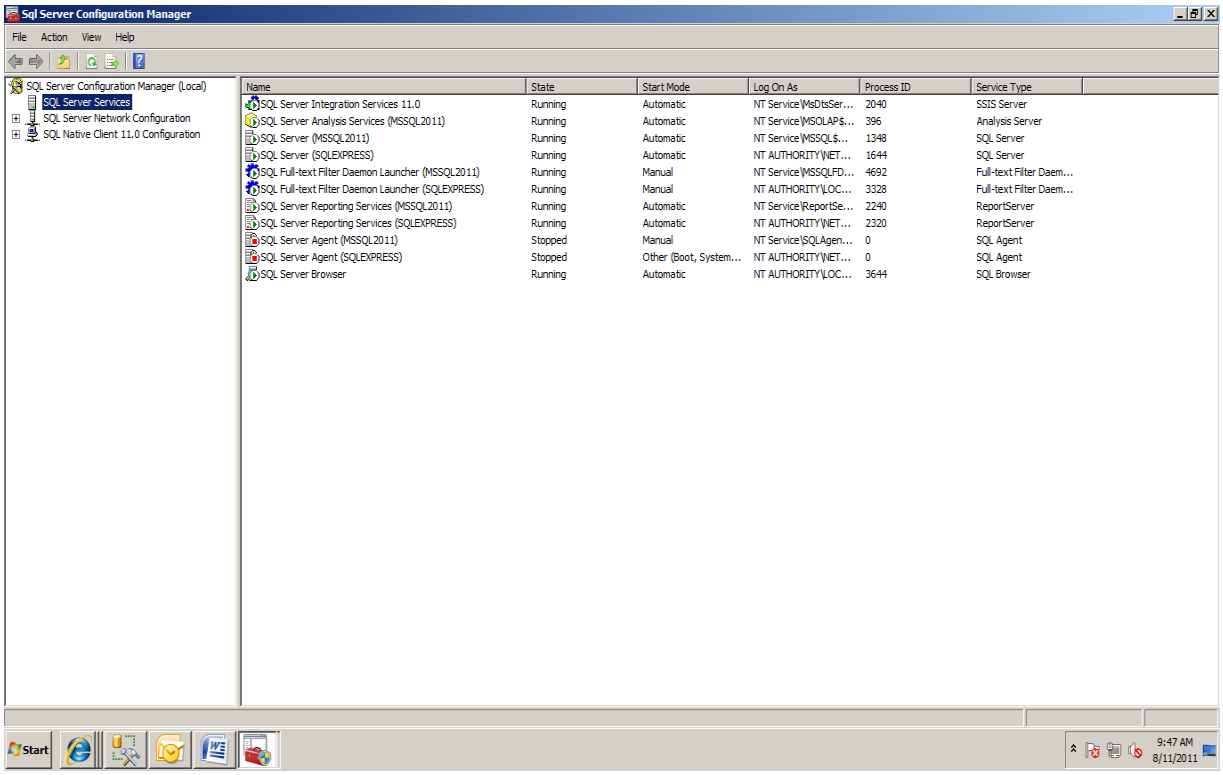
Increases the number of extents that are allocated for each file in a filegroup. This option may be helpful for data warehouse applications that have a limited number of users running index or data scans. It should not be used in other applications because it might adversely affect performance. This option is not supported in 32-bit releases of SQL Server.

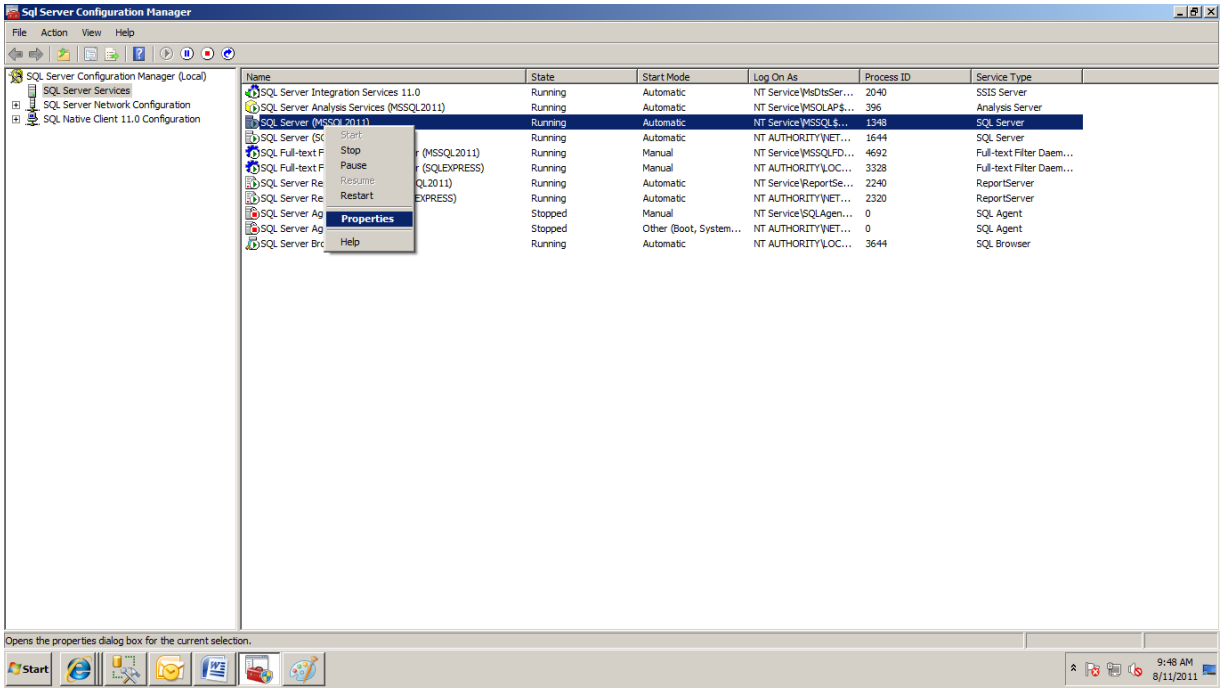
4.2 Compatibility Support

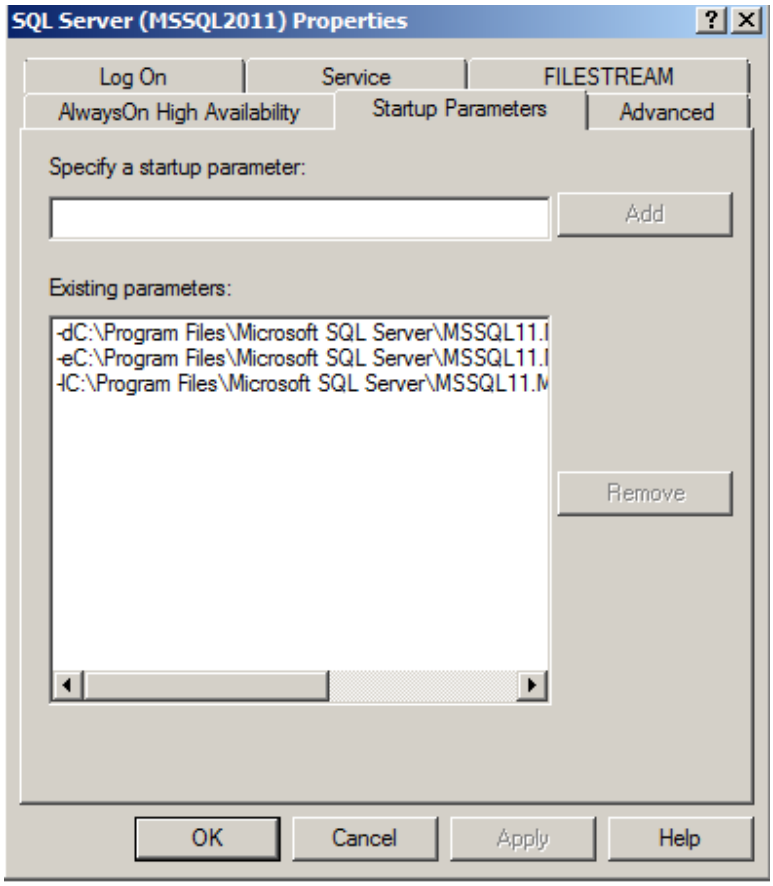
The **-h** parameter is not supported in Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3). This parameter was used in earlier versions of 32-bit instances of SQL Server to reserve virtual memory address space for Hot Add memory metadata when AWE is enabled.

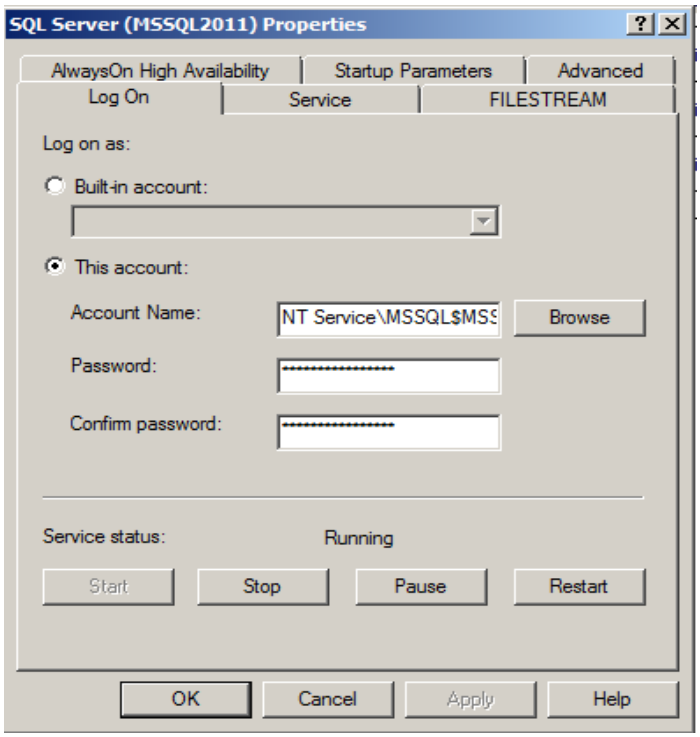
4.3 How to Change Startup Parameter in SQL Server Denali

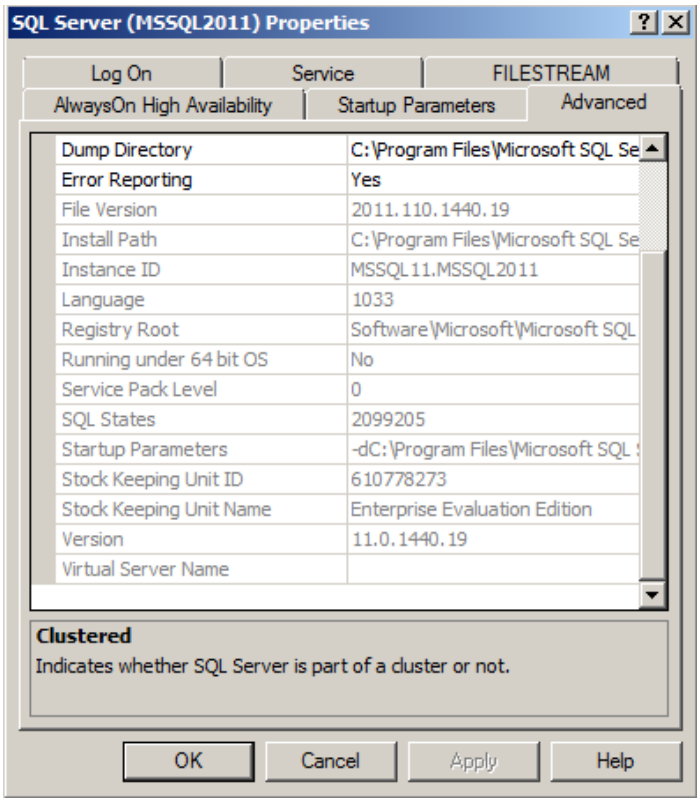
1. Open SQL Configuration Manager
2. Click on SQL Server Services
3. Select the SQL Server Database Services and Right Click
4. In Properties box edit the parameters as shown below


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration
SQL Native Client 11.0 Configuration
Name State Start Mode Log On As Process ID Service Type
SQL Server Integration Services 11.0 Running Automatic NT Service\MsDtsSer... 2040 SSIS Server
SQL Server Analysis Services (MSSQL2011) Running Automatic NT Service\MSOLAP$... 396 Analysis Server
SQL Server (MSSQL2011) Running Automatic NT Service\MSSQL$... 1348 SQL Server
SQL Server (SQLEXPRESS) Running Automatic NT AUTHORITY\NET... 1644 SQL Server
SQL Full-text Filter Daemon Launcher (MSSQL2011) Running Manual NT Service\MSSQLFD... 4692 Full-text Filter Daem...
SQL Full-text Filter Daemon Launcher (SQLEXPRESS) Running Manual NT AUTHORITY\LOC... 3328 Full-text Filter Daem...
SQL Server Reporting Services (MSSQL2011) Running Automatic NT Service\ReportSe... 2240 ReportServer
SQL Server Reporting Services (SQLEXPRESS) Running Automatic NT AUTHORITY\NET... 2320 ReportServer
SQL Server Agent (MSSQL2011) Stopped Manual NT Service\SQLAgen... 0 SQL Agent
SQL Server Agent (SQLEXPRESS) Stopped Other (Boot, System... NT AUTHORITY\NET... 0 SQL Agent
SQL Server Browser Running Automatic NT AUTHORITY\LOC... 3644 SQL Browser
Start
9:47 AM
8/11/2011


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration
SQL Native Client 11.0 Configuration
Name State Start Mode Log On As Process ID Service Type
SQL Server Integration Services 11.0 Running Automatic NT Service\MsDtsSer... 2040 SSIS Server
SQL Server Analysis Services (MSSQL2011) Running Automatic NT Service\MSOLAP$... 396 Analysis Server
SQL Server (MSSQL2011) Running Automatic NT Service\MSSQL$... 1348 SQL Server
SQL Server (SQLEXPRESS) Running Automatic NT AUTHORITY\NET... 1644 SQL Server
SQL Full-text Filter Daemon Launcher (MSSQL2011) Running Manual NT Service\MSSQLFD... 4692 Full-text Filter Daem...
SQL Full-text Filter Daemon Launcher (SQLEXPRESS) Running Manual NT AUTHORITY\LOC... 3328 Full-text Filter Daem...
SQL Server Reporting Services (MSSQL2011) Running Automatic NT Service\ReportSe... 2240 ReportServer
SQL Server Reporting Services (SQLEXPRESS) Running Automatic NT AUTHORITY\NET... 2320 ReportServer
SQL Server Agent (MSSQL2011) Stopped Manual NT Service\SQLAgen... 0 SQL Agent
SQL Server Agent (SQLEXPRESS) Stopped Other (Boot, System... NT AUTHORITY\NET... 0 SQL Agent
SQL Server Browser Running Automatic NT AUTHORITY\LOC... 3644 SQL Browser
Start
Stop
Pause
Resume
Restart
Properties
Help
Opens the properties dialog box for the current selection.
9:48 AM
8/11/2011


SQL Server (MSSQL2011) Properties
Log On
Service
FILESTREAM
AlwaysOn High Availability
Startup Parameters
Advanced
Specify a startup parameter:
Add
Existing parameters:
-dC:\Program Files\Microsoft SQL Server\MSSQL11.I
-eC:\Program Files\Microsoft SQL Server\MSSQL11.I
-lC:\Program Files\Microsoft SQL Server\MSSQL11.M
Remove
OK
Cancel
Apply
Help


SQL Server (MSSQL2011) Properties
AlwaysOn High Availability
Startup Parameters
Advanced
Log On
Service
FILESTREAM
Log on as:
Built-in account:
This account:
Account Name:
NT Service\MSSQL$MSS
Browse
Password:
Confirm password:
Service status:
Running
Start
Stop
Pause
Restart
OK
Cancel
Apply
Help


SQL Server (MSSQL2011) Properties
Log On
Service
FILESTREAM
AlwaysOn High Availability
Startup Parameters
Advanced
Dump Directory
C:\Program Files\Microsoft SQL Se
Error Reporting
Yes
File Version
2011.110.1440.19
Install Path
C:\Program Files\Microsoft SQL Se
Instance ID
MSSQL11.MSSQL2011
Language
1033
Registry Root
Software\Microsoft\Microsoft SQL
Running under 64 bit OS
No
Service Pack Level
0
SQL States
2099205
Startup Parameters
-dC:\Program Files\Microsoft SQL
Stock Keeping Unit ID
610778273
Stock Keeping Unit Name
Enterprise Evaluation Edition
Version
11.0.1440.19
Virtual Server Name
Clustered
Indicates whether SQL Server is part of a cluster or not.
OK
Cancel
Apply
Help

5 Code Snippets

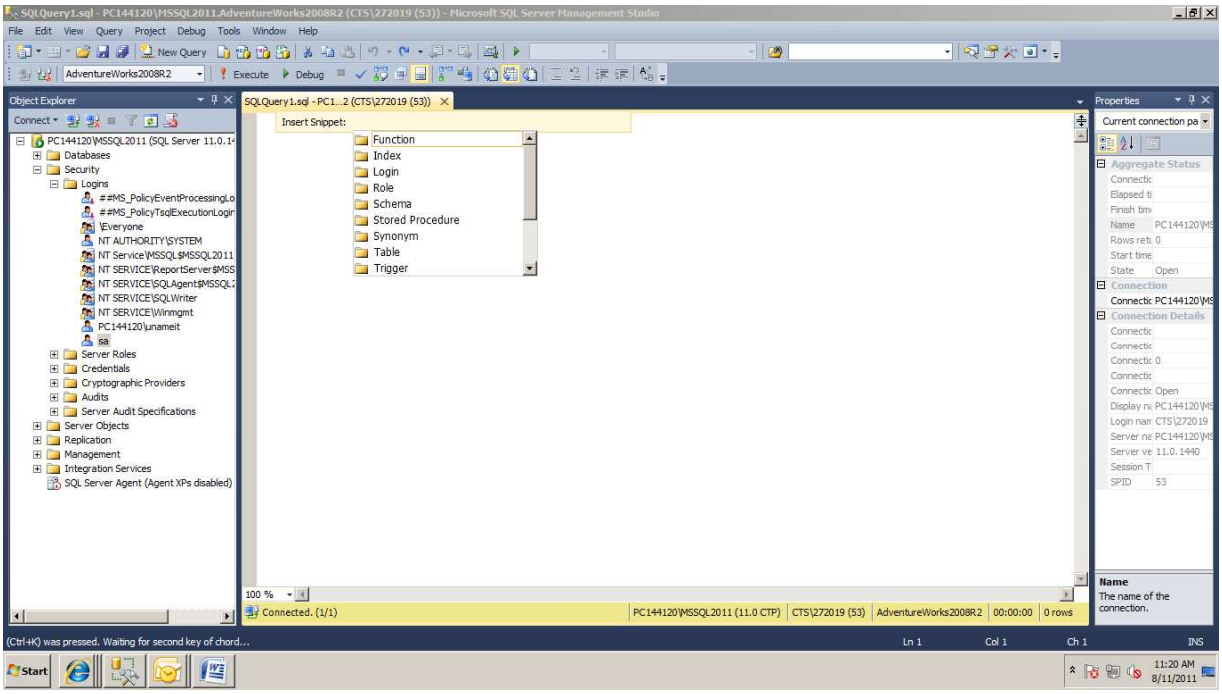
A Transact-SQL code snippet is a template you can use as a starting point when writing new Transact-SQL statements in the Database Engine Query Editor. Transact-SQL snippets contain replacement points: text that suggests the syntax relevant to that point. For example, the CREATE TABLE snippet has replacement points for elements such as the table name, column names, and column data types. After inserting the snippet, you must change the replacement text to form a valid Transact-SQL statement.

5.1 Completing the Snippet Statement

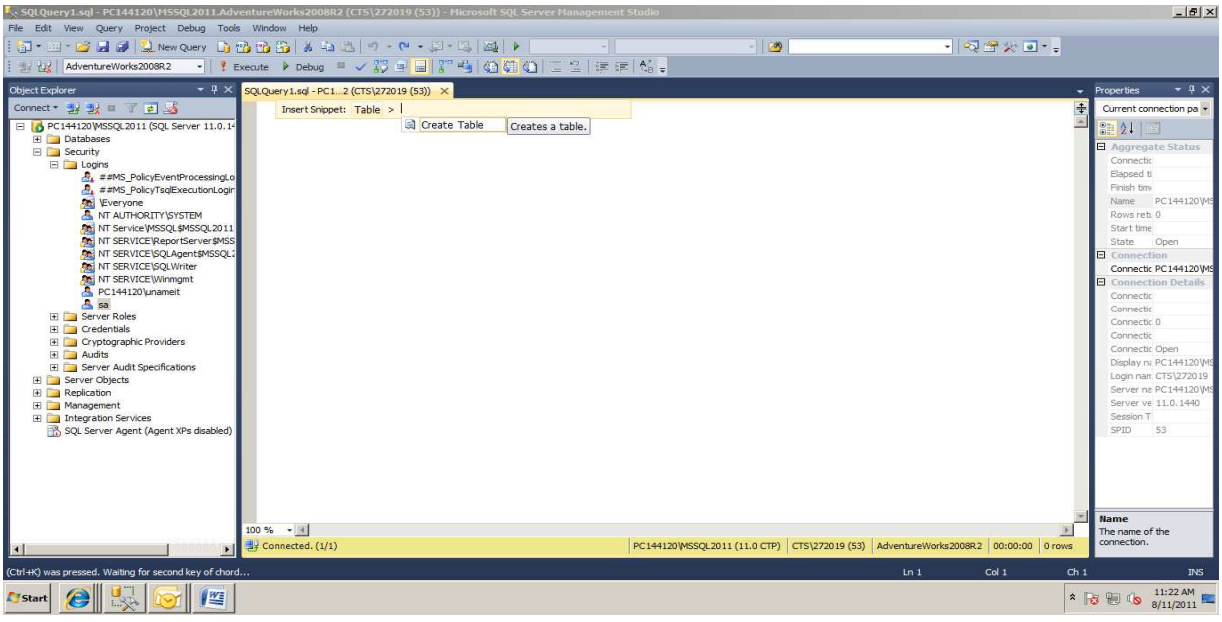
1. Use the TAB key to move from one replacement point to the next. Use SHIFT+TAB to move to the previous replacement point.
2. Click CTRL+SPACE to invoke IntelliSense.
3. Select an item from the list, or type a replacement of your choice.

5.2 Inserting Code Snippets

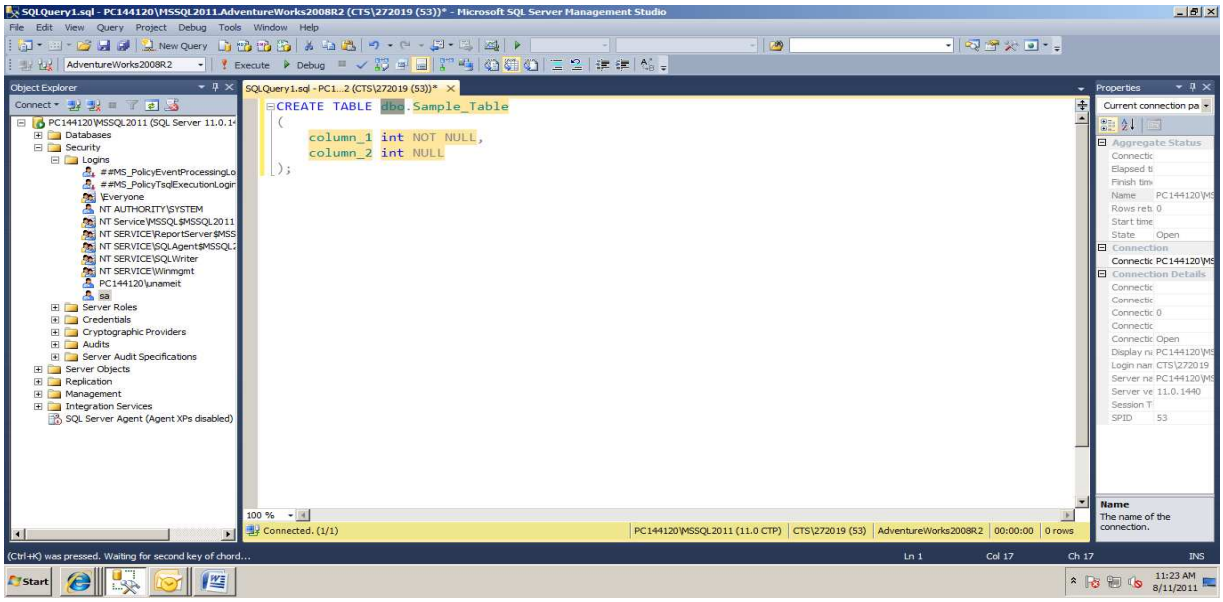
To insert code snippets, right click on the query window or press CTRL+K followed by CTRL+X and click on the "Insert Snippet..." menu item. When you click on the "Insert Snippet..." menu item, you will see a snippet picker with several options like shown below.



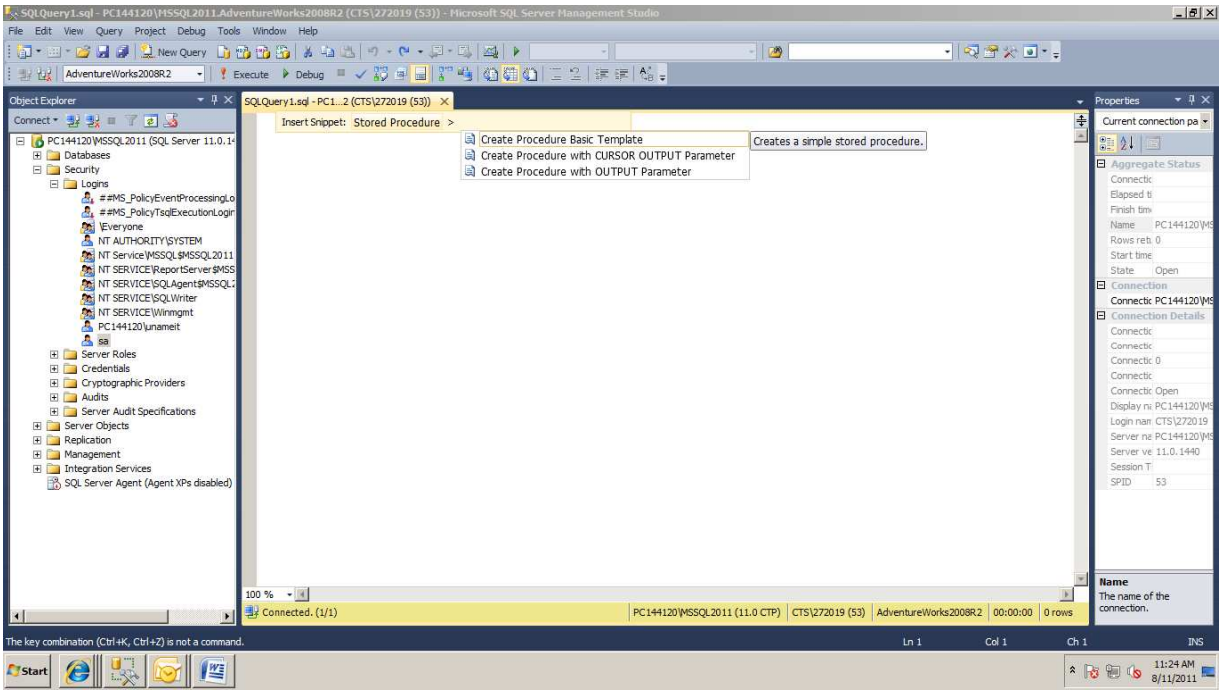
In order to create a table, choose the Create Table option from the snippet picker and click on it as shown below:



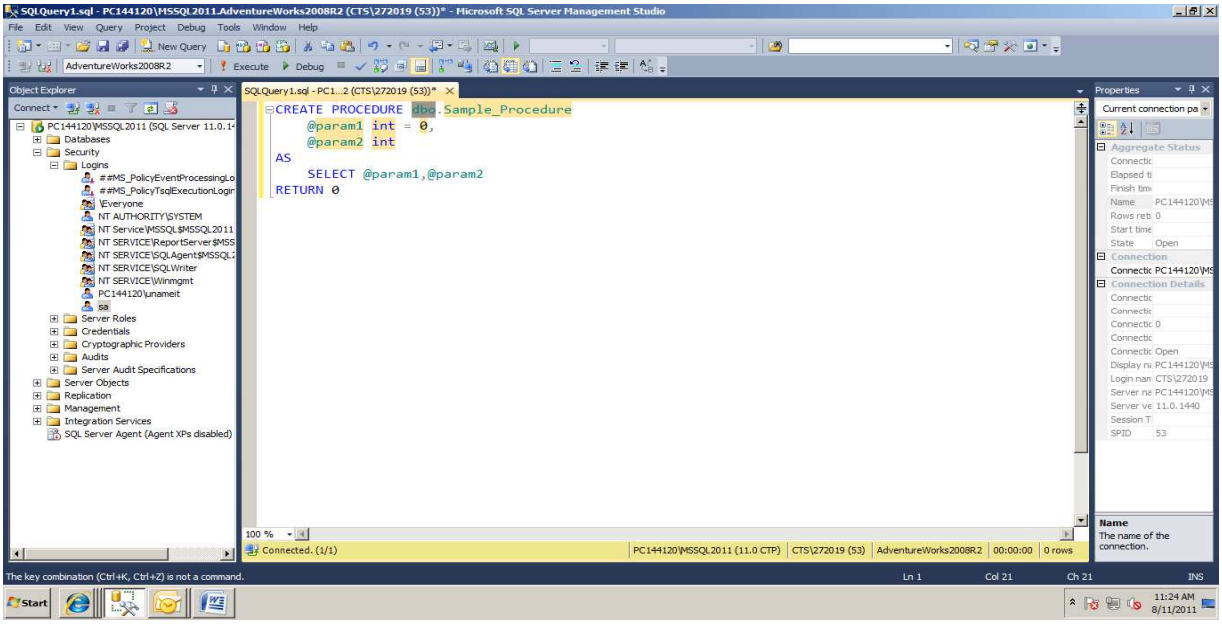
It will bring up the table creation script template in the query window as you can see below; you can now modify it as per your need and execute it to create the table.



A category might also have multiple script templates (code snippets) as you can see below for Stored Procedure. Here we have three different templates to create a stored procedure, one is basic, the second is with a CURSOR as an output parameter and third is with an OUTPUT parameter.



The basic template snippet creates a new stored procedure



5.3 Insert Surround-with Transact-SQL snippets

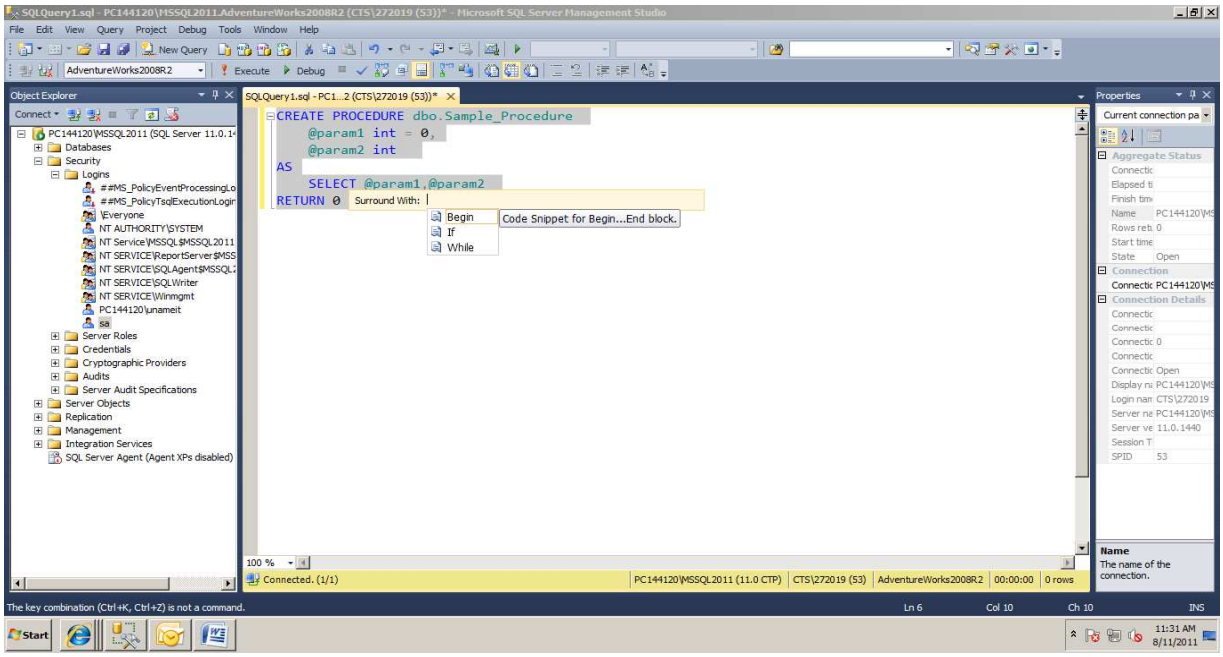
A surround-with snippet is a template you can use as a starting point when enclosing a set of Transact-SQL statements in a BEGIN, IF, or WHILE block

5.3.1 To insert a surround-with snippet

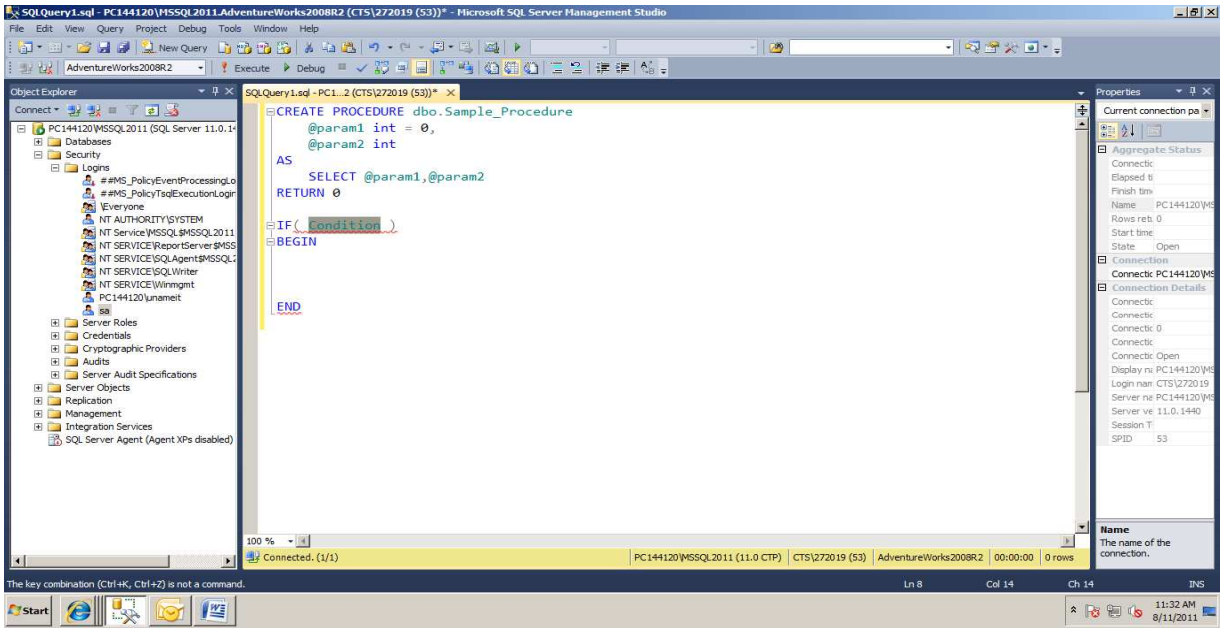
1. In the Database Engine Query Editor window, select the set of statements to be included in the block.
2. Use one of these three methods to display the list of surround-with snippets:
   - Type CTRL+K, CTRL+S.
   - From the Edit menu, point to IntelliSense, and then select the Surround With command.
   - Right-click the selected text, and then select the Surround With command from the context menu.
3. Select the name of the snippet (BEGIN, IF, or WHILE) from the list either by using the mouse, or by typing the name of the snippet and pressing TAB or ENTER.

5.3.2 BEGIN, WHILE and IF Blocks

Not only can insert script templates, but you can also surround your code with either BEGIN, WHILE or IF blocks. To surround lines of code, select one or more lines of code and right click or hit CTRL+K followed by CTRL+S and then click on the "Surround With..." menu item as shown below.

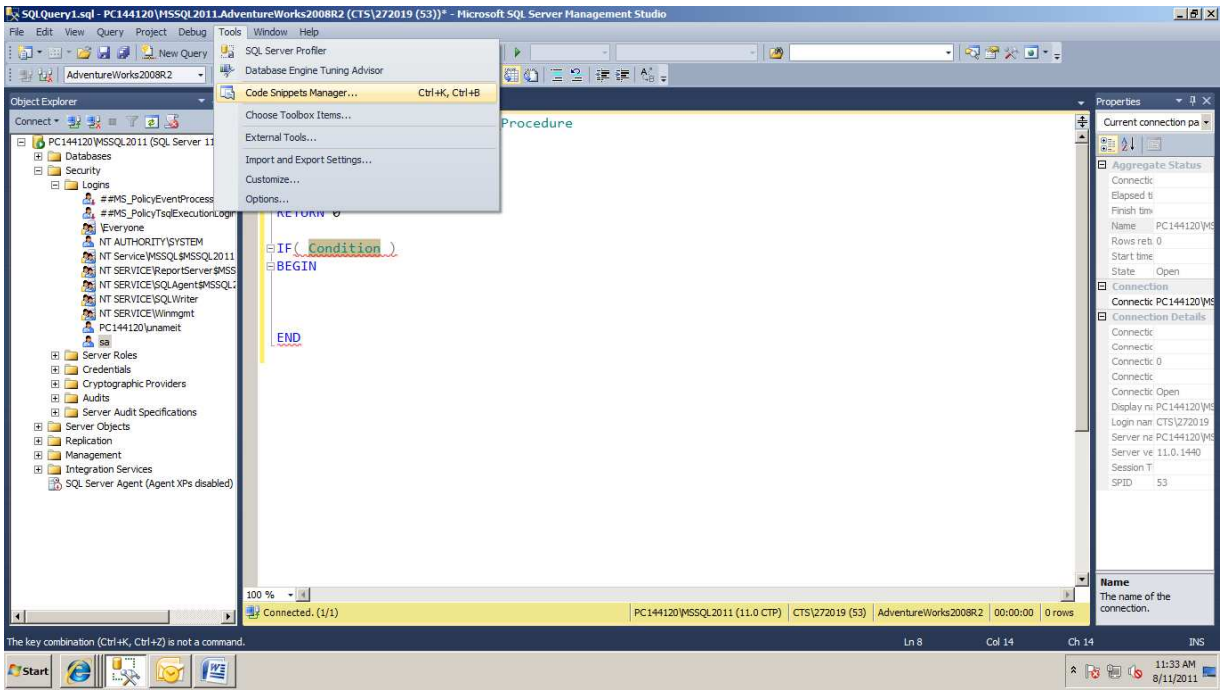


In the below scenario lines of code are surrounded with an IF block.

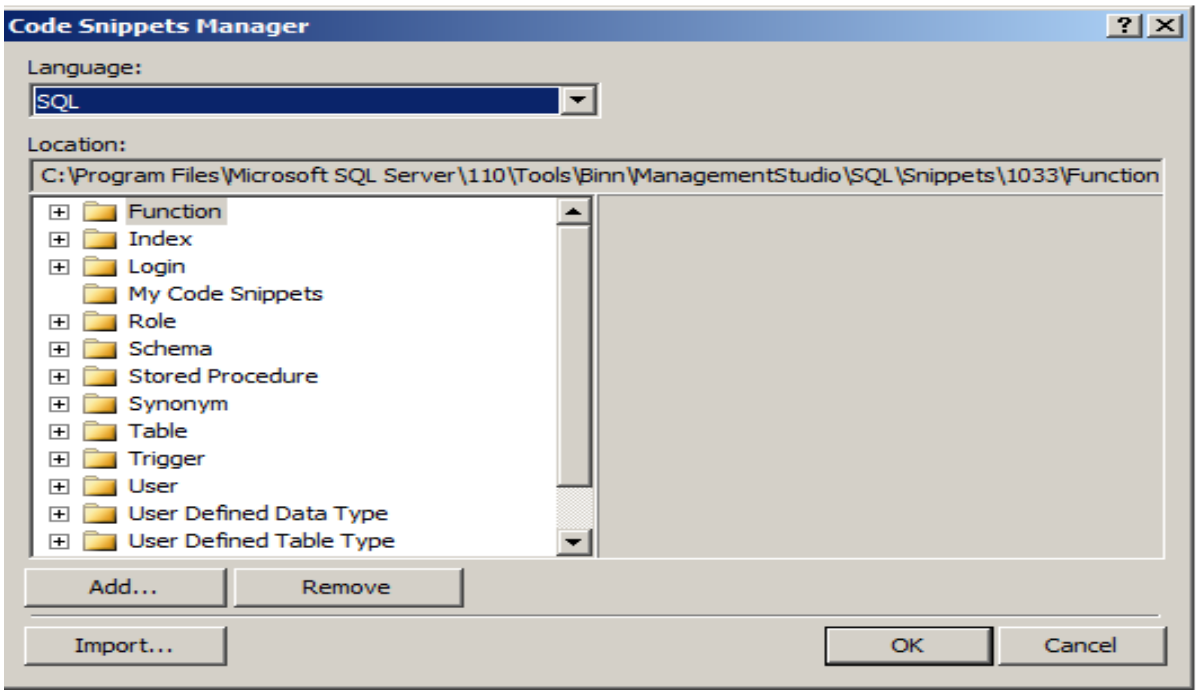


5.4 Snippet Management

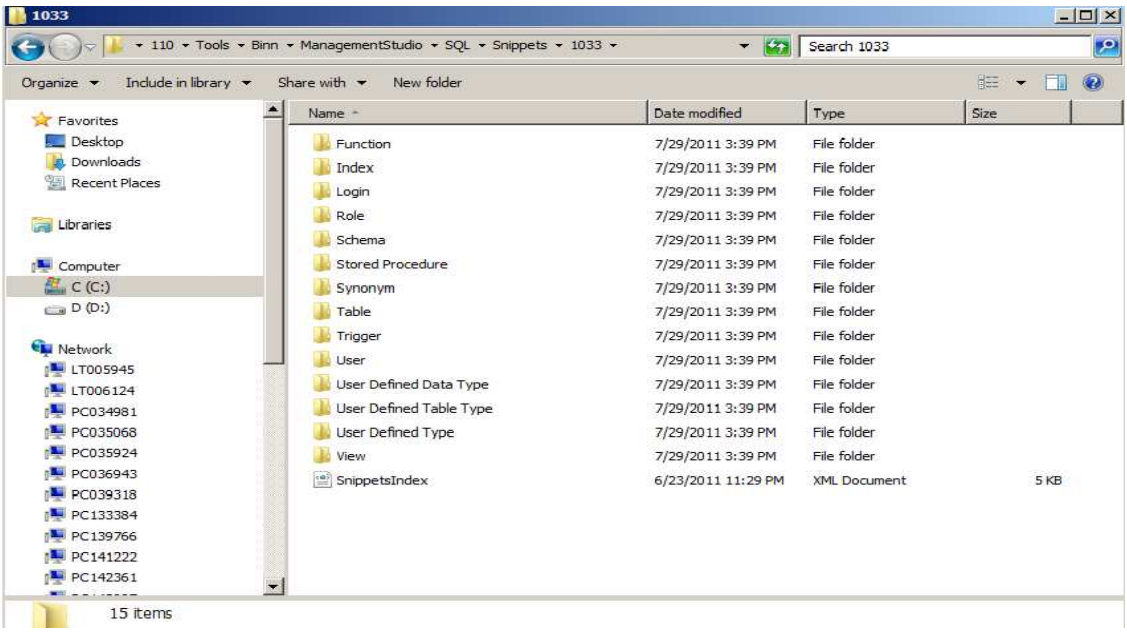
Code Snippets Manager manages the snippet code; to launch it, go to Tools in SSMS then click on "Code Snippets Manager..." as shown below or hit CTRL+K followed by CTRL+B.



With the Code Snippets Manager we can view all available templates (code snippets), add, remove and import. We can modify and add templates to suit your requirements:



All the templates are stored on the file system as XML files in the folder : **<install drive>\Program Files (x86)\Microsoft SQL Server\110\Tools\Binn\ManagementStudio\SQL\Snippets\1033**. Each group is categorized and stored in different folders. For example the Table folder is shown below.



6 Juneau

SQL Server Development Tools (SSDT) - 'Juneau' - brings testing, debugging, version control, refactoring, dependency checking, and deployment to database development in Visual Studio. It was previewed recently along with SQL Server 'Denali' CTP.

Some of the features in the preview are -

- New Testing and Debugging method which works with a local database that's created automatically for each database project
- Schema Compare – for comparing a SQL Server Project Schema to a target database and highlight differences
- Publish Database feature, that supports both onsite SQL Server as well as SQL Azure
- Declarative database development through Power Buffer
- Enhanced TSQL editor with features like IntelliSense, Go To Definition and Find All References, Refactor contextual menu and more, similar to C# and VB editors in Visual Studio

SQL Server Developer Tools finally gives application developers what they need to get everything done without ever leaving Visual Studio—and it delivers many powerful new capabilities as well.

6.1 Features of JUNEAU:

6.1.1 Model-Based Development

The key concept in SSDT is that it uses a model-based approach. That is, there is always an in-memory representation of what a database looks like, and all the SSDT tools (designers, validations, IntelliSense, schema compare, etc.) operate on that model. This model can be backed by a live database (on-premises or Azure), an offline database project, or a snapshot taken of an offline database project at any point in time. But to reiterate, the tools are agnostic to the model's backing, and work exclusively against the model itself. Thus, we get consistent experience in any scenario—regardless of whether working with on-premises or cloud databases, offline projects, or versioned model snapshots.

6.1.2 SSMS in Server Explorer

The Server Explorer now provides most of the functionality that developers are accustomed to in SSMS. Although the experience is strikingly similar when working against a connected database, remember that SSDT tools only operate on a model. So when working connected, SSDT is actually creating a model from the database—on the fly—and then allowing you to edit that model.

When you "save" a schema change, SSDT works out the necessary script to update the database to reflect the change made to the model. Of course, the end result is the same as the

connected SSMS experience, so it isn't strictly necessary to understand the. It's just that when you're working in Server Explorer, the backing happens to be a live connected database.

You can also open query windows to compose and execute T-SQL statements directly against the database. Overall, these Server Explorer-based features give you the same connected management experience you are accustomed to in SSMS; which is an imperative, script-based approach to directly managing a stateful database.

6.1.3 Offline Development in Visual Studio

SSDT offers offline experience with the new SQL Server project type in Visual Studio 2010. The T-SQL script files in a SQL Server project are declarative in nature (all ***CREATE*** statements; no ***ALTER*** statements), which is an entirely different approach to how you're accustomed to "developing" databases in SSMS against a live stateful database. Essentially, you get to focus on "this is how the database should look," and let the tool worry about and figure out the appropriate T-SQL statements to actually update the live database to match your definition.

The SQL Server project enjoys all the same capabilities of any Visual Studio project. This includes not only source control, but all the common code navigation and refactoring paradigms (like Rename, Goto Definition, and Find All References) that have come to be accepted as indispensible tools of the modern IDE. And the model's rich metadata provides a far better IntelliSense than what SSMS has been offering since SQL Server 2008. Also breakpoints can be set, single-step through T-SQL code, and utilize the Locals window in Visual Studio much like debugging .NET project types. With SSDT, application and database development tooling has now finally been unified under one roof: Visual Studio 2010.

With the model-based approach we can generate models from almost anything. As already explained, when connected directly via Server Explorer, SSDT creates a model from the connected database. When creating a SQL Server project (which can be imported from any existing database, script, or snapshot), we are creating an offline source-controlled project inside Visual Studio that fully describes a database. Now SSDT generates a model backed by project, and so the design experience offline is just the same as when connected; the designers, IntelliSense, validation checks, and all other SSDT features all work exactly the same way.

SSDT also provides a new local database runtime for testing and debugging. This is a lightweight, single user instance of SQL Server that spins up on demand when you build your SQL Server project. This is a great way to test your offline work before deploying to a live server or the cloud.

6.1.4 Versioning and Snapshots

A database project gives you an offline definition of a database. Like all projects, each database object (table, view, stored procedure) lives as a source file that can be placed under version control. Thus, the project system enables you to preserve and protect the definition of the database, as opposed to having the definition live within the database itself.

At any point in time, and as often as you'd like, you can create a database ***snapshot***. A snapshot is nothing more than a persisted serialized version of the model based on the current project at the time the snapshot is taken. As such, you can use SSDT to develop, deploy, and synchronize database structures across local/cloud databases and differently versioned offline database projects.

6.1.5 Targeting SQL Azure

SSDT projects have a target switch that lets you specify which SQL Server edition or platform the project is intended to be deployed to. All the validation checks against the project-backed model are based on this setting, so it's simply a matter of choosing SQL Azure as the target to ensure that your database can be deployed to the cloud without any problems. If your database project defines something that is not supported in Azure (a table with no clustered index, for example), it will get flagged as an error automatically when you attempt to publish.

7 Over Clause Support Enhanced

The OVER clause has been extended to support window functions. Window functions perform a calculation across a set of rows that are in some relationship to the current row.

- ROWS or RANGE clause can be used over a set of rows to calculate a moving average or cumulative total.
- Ordering rows within a partition is now supported in the aggregate functions that allow the OVER clause to be specified.
- Determines the partitioning and ordering of a rowset before the associated window function is applied.
- OVER clause can be used with functions to compute aggregated values such as moving averages, cumulative aggregates, running totals, or a top N per group results.

Syntax:

```
OVER (
       [ <PARTITION BY clause> ]
       [ <ORDER BY clause> ]
       [ <ROW or RANGE clause> ]
      )

<PARTITION BY clause> ::=
PARTITION BY value_expression , ... [ n ]

<ORDER BY clause> ::=
ORDER BY order_by_expression
    [COLLATE collation_name ]
    [ ASC | DESC ]
    [ ,...n ]
```

For detailed syntax; refer http://msdn.microsoft.com/en-us/library/ms189461(v=sql.110).aspx

- PARTITION BY divides the query result set into partitions. The window function is applied to each partition separately and computation restarts for each partition.

- Value_expression Specifies the column by which the rowset is partitioned. value_expression can only refer to columns made available by the FROM clause. value_expression cannot refer to

expressions or aliases in the select list. value_expression can be a column expression, scalar subquery, scalar function, or user-defined variable.

- ORDER BY clause defines the logical order of the rows within each partition of the result set. That is, it specifies the logical order in which the window functioncalculation is performed.

- ORDER_BY_EXPRESSION specifies a column or expression on which to sort. Order_by_expression can only refer to columns made available by the FROM clause. An integer cannot be specified to represent a column name or alias.

- COLLATE Collation_name Specifies that the ORDER BY operation should be performed according to the collation specified in Collation_name. Collation_name can be either a Windows collation name or a SQL collation name. COLLATE is applicable only for columns of type char, varchar, nchar, and nvarchar.

- ASC | DESC specifies that the values in the specified column should be sorted in ascending or descending order. ASC is the default sort order. Null values are treated as the lowest possible values.

- ROWS | RANGE further limits the rows within the partition by specifying start and end points within the partition. This is done by specifying a range of rows with respect to the current row either by logical association or physical association. Physical association is achieved by using the ROWS clause.

- The ROWS clause limits the rows within a partition by specifying a fixed number of rows preceding or following the current row.

- The RANGE clause logically limits the rows within a partition by specifying a range of values with respect to the value in the current row.

7.1 Using the OVER clause with the ROW_NUMBER function

The following example shows using the OVER clause with ROW_NUMBER function to display a row number for each row within a partition. The ORDER BY clause specified in the OVER clause orders the rows in each partition by the column SalesYTD. The ORDER BY clause in the SELECT statement determines the order in which the entire query result set is returned.

```
USE AdventureWorks2008R2;
GO
SELECT ROW_NUMBER () OVER (PARTITION BY PostalCode ORDER BY SalesYTD
DESC) AS 'Row Number',
    p.LastName, s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson AS s
    INNER JOIN Person.Person AS p
        ON s.BusinessEntityID = p.BusinessEntityID
    INNER JOIN Person.Address AS a
        ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0
ORDER BY PostalCode;
GO
```

7.2 Using the OVER clause with aggregate functions

The following example uses the OVER clause with aggregate functions over all rows returned by the query. In this example, using the OVER clause is more efficient than using subqueries to derive the aggregate values.

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, ProductID, OrderQty
    ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total'
    ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Avg'
    ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Count'
    ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Min'
    ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Max'
FROM Sales.SalesOrderDetail
WHERE SalesOrderID IN(43659,43664);
GO
```

7.3 Using the OVER clause with an aggregate function in a calculated value

```
USE AdventureWorks2008R2;
GO
SELECT SalesOrderID, ProductID, OrderQty
    ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total'
    ,CAST(1. * OrderQty / SUM(OrderQty) OVER(PARTITION BY SalesOrderID)
        *100 AS DECIMAL(5,2))AS 'Percent by ProductID'
FROM Sales.SalesOrderDetail
WHERE SalesOrderID IN(43659,43664);
```

7.4 Using the AVG and SUM functions with the OVER clause to provide a moving average and cumulative total

The following example uses the AVG and SUM functions with the OVER clause to provide a moving average and cumulative total of yearly sales for each territory in the Sales.SalesPerson table. The data is partitioned by TerritoryID and logically ordered by SalesYTD. This means that the AVG function is computed for each territory based on the sales year. Notice that for TerritoryID 1, there are two rows for sales year 2005 representing the two sales people with sales that year. The average sales for these two rows are computed and then the third row representing sales for the year 2006 is included in the computation.

```
USE AdventureWorks2008R2;
GO
SELECT BusinessEntityID, TerritoryID
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SalesYTD,1) AS  SalesYTD
    ,CONVERT(varchar(20),AVG(SalesYTD) OVER (PARTITION BY TerritoryID
                                             ORDER BY
DATEPART(yy,ModifiedDate)
                                            ),1) AS MovingAvg
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
                                             ORDER BY
DATEPART(yy,ModifiedDate)
                                             ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
ORDER BY TerritoryID,SalesYear;
```

7.5 Specifying the ROWS clause

The following example uses the ROWS clause to define a window over which the rows are computed as the current row and the ***N*** number of rows that follow (1 row in this example).

```
SELECT BusinessEntityID, TerritoryID
    ,CONVERT(varchar(20),SalesYTD,1) AS  SalesYTD
    ,DATEPART(yy,ModifiedDate) AS SalesYear
    ,CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
                                             ORDER BY
DATEPART(yy,ModifiedDate)
                                             ROWS BETWEEN CURRENT ROW AND 1
FOLLOWING ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5;
```

7.6 Limitations and Restrictions

- The OVER clause cannot be used with the CHECKSUM aggregate function.
- RANGE cannot be used with <unsigned value specification> PRECEDING or <unsigned value specification> FOLLOWING.
- Depending on the ranking, aggregate, or analytic function used with the OVER clause, <ORDER BY clause> and/or the <ROWS and RANGE clause> may not be supported.

8 Dynamic Management Views and Functions

The following system views have been added or modified:

8.1 sys.dm_exec_query_stats (Transact-SQL)

Added four columns to help troubleshoot long running queries. You can use the total_rows, min_rows, max_rows and last_rows aggregate row count columns to separate queries that are returning a large number of rows from problematic queries that may be missing an index or have a bad query plan.

8.2 sys.dm_os_volume_stats (Transact-SQL)

This dynamic management functions returns information about the operating system volume (directory) on which the specified databases and files are stored. Use this dynamic management function to check the attributes of the physical disk drive or return available free space information about the directory.

8.3 sys.dm_os_windows_info (Transact-SQL)

This dynamic management view returns one row that displays Windows operating system version information such as the OS version or language ID.

8.4 sys.dm_server_memory_dumps (Transact-SQL)

Returns one row for each memory dump file generated by the SQL Server Database Engine. Use this dynamic management view to troubleshoot potential issues.

8.5 sys.dm_server_services (Transact-SQL)

Returns information about the SQL Server, Full-Text, and SQL Server Agent services in the current instance of SQL Server. Use this dynamic management view to report status information about these services.

8.6 sys.dm_server_registry (Transact-SQL)

Returns configuration and installation information that is stored in the Windows registry for the current instance of SQL Server. Returns one row per registry key. Use this dynamic management view to return information such as the SQL Server services that are available on the host machine or network configuration values for the instance of SQL Server.

9 Ad-Hoc Query Paging

The Order By option in the SQL SELECT statement has been enhanced in SQL Server 2011. A combination of OFFSET and FETCH along with ORDER BY gives control of paging through a result set. Using this technique can really help performance by bring back only the results needed to show users when they are needed.

OFFSET {*integer_constant* | *offset_row_count_expression*} {ROW | ROWS}

- Specifies the number of rows to skip before it starts to return rows from the query expression. The value can be an integer constant or expression that is greater than or equal to zero.
- Offset_row_count_expression can be a variable, parameter, or constant scalar subquery. When a subquery is used, it cannot reference any columns defined in the outer query scope. That is, it cannot be correlated with the outer query.
- ROW and ROWS are synonyms and are provided for ANSI compatibility.
- In query execution plans, the offset row count value is displayed in the Offset attribute of the TOP query operator.

FETCH {FIRST | NEXT} {*integer_constant* | *fetch_row_count_expression*} {ROW | ROWS} ONLY

- Specifies the number of rows to return after the OFFSET clause has been processed. The value can be an integer constant or expression that is greater than or equal to one.
- *Fetch_row_count_expression* can be a variable, parameter, or constant scalar subquery. When a subquery is used, it cannot reference any columns defined in the outer query scope. That is, it cannot be correlated with the outer query.
- FIRST and NEXT are synonyms and are provided for ANSI compatibility.
- ROW and ROWS are synonyms and are provided for ANSI compatibility.
- In query execution plans, the offset row count value is displayed in the Rows or Top attribute of the TOP query operator.

The ORDER BY clause is not valid in views, inline functions, derived tables, and subqueries, unless either the TOP or OFFSET and FETCH clauses are also specified. When ORDER BY is used in these objects, the clause is used only to determine the rows returned by the TOP clause or OFFSET and FETCH clauses. The ORDER BY clause does not guarantee ordered results when these constructs are queried, unless ORDER BY is also specified in the query itself.

Using OFFSET and FETCH in a view does not change the updateability property of the view.
OFFSET and FETCH are not supported in indexed views or in a view that is defined by using the CHECK OPTION clause.

OFFSET and FETCH can be used in any query that allows TOP and ORDER BY with the following limitations:

- The OVER clause does not support OFFSET and FETCH.
- OFFSET and FETCH cannot be specified directly in INSERT, UPDATE, MERGE, and DELETE statements, but can be specified in a subquery defined in these statements. For example, in the INSERT INTO SELECT statement, OFFSET and FETCH can be specified in the SELECT statement.
- In a query that uses UNION, EXCEPT or INTERSECT operators, OFFSET and FETCH can only be specified in the final query that specifies the order of the query results.
- TOP cannot be combined with OFFSET and FETCH in the same query expression (in the same query scope).

9.1 Using OFFSET and FETCH to limit the rows returned

The OFFSET and FETCH clauses can be used instead of the TOP clause to implement a query paging solution and limit the number of rows sent to a client application.

Using OFFSET and FETCH as a paging solution requires running the query one time for each "page" of data returned to the client application. For example, to return the results of a query in 10-row increments, you must execute the query one time to return rows 1 to 10 and then run the query again to return rows 11 to 20 and so on. Each query is independent and not related to each other in any way. This means that, unlike using a cursor in which the query is executed once and state is maintained on the server, the client application is responsible for tracking state. To achieve stable results between query requests using OFFSET and FETCH, the following conditions must be met:

1. The underlying data that is used by the query must not change. That is, either the rows touched by the query are not updated or all requests for pages from the query are executed in a single transaction using either snapshot or serializable transaction isolation.
2. The ORDER BY clause contains a column or combination of columns that are guaranteed to be unique.

Examples

The following examples use OFFSET and FETCH to limit the number of rows returned by a query.

A. Specifying integer constants for OFFSET and FETCH values

The following example specifies an integer constant as the value for the OFFSET and FETCHES clauses. The first query returns all rows sorted by the column DepartmentID. Compare the results returned by this query with the results of the two queries that follow it. The next query uses the clause OFFSET 5 ROWS to skip the first 5 rows and return all remaining rows. The final query uses the clause OFFSET 0 ROWS to start with the first row and then uses FETCH NEXT 10 ROWS ONLY to limit the rows returned to 10 rows from the sorted result set.

```
USE AdventureWorks2008R2;
GO
-- Return all rows sorted by the column DepartmentID.
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID;

-- Skip the first 5 rows from the sorted result set and return all remaining
rows.
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID OFFSET 5 ROWS;

-- Skip 0 rows and return only the first 10 rows from the sorted result set.
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID
    OFFSET 0 ROWS
    FETCH NEXT 10 ROWS ONLY;
```

B. Specifying variables for OFFSET and FETCH values

The following example declares the variables @StartingRowNumber and @FetchRows and specifies these variables in the OFFSET and FETCH clauses.

```
USE AdventureWorks2008R2;
GO
-- Specifying variables for OFFSET and FETCH values
DECLARE @StartingRowNumber tinyint = 1
      , @FetchRows tinyint = 8;
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID ASC
    OFFSET @StartingRowNumber ROWS
    FETCH NEXT @FetchRows ROWS ONLY;
```

C. Specifying expressions for OFFSET and FETCH values

The following example uses the expression @StartingRowNumber - 1 to specify the OFFSET value and the expression @EndingRowNumber - @StartingRowNumber + 1 to specify the FETCH value. In addition, the query hint, OPTIMIZE FOR, is specified. This hint can be used to provide a particular value for a local variable when the query is compiled and optimized. The value is used only during query optimization, and not during query execution.

```
USE AdventureWorks2008R2;
GO
-- Specifying expressions for OFFSET and FETCH values
DECLARE @StartingRowNumber tinyint = 1
      , @EndingRowNumber tinyint = 8;
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID ASC
    OFFSET @StartingRowNumber - 1 ROWS
    FETCH NEXT @EndingRowNumber - @StartingRowNumber + 1 ROWS ONLY
OPTION ( OPTIMIZE FOR (@StartingRowNumber = 1, @EndingRowNumber = 20) );
```

D. Specifying a constant scalar subquery for OFFSET and FETCH values

The following example uses a constant scalar subquery to define the value for the FETCH clause. The subquery returns a single value from the column PageSize in the table dbo.AppSettings.

```
-- Specifying a constant scalar subquery
USE AdventureWorks2008R2;
GO
CREATE TABLE dbo.AppSettings (AppSettingID int NOT NULL, PageSize int NOT
NULL);
GO
INSERT INTO dbo.AppSettings VALUES(1, 10);
GO
DECLARE @StartingRowNumber tinyint = 1;
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID ASC
    OFFSET @StartingRowNumber ROWS
    FETCH NEXT (SELECT PageSize FROM dbo.AppSettings WHERE AppSettingID = 1)
ROWS ONLY;
```

E. Running multiple queries in a single transaction

The following example shows one method of implementing a paging solution that ensures stable results are returned in all requests from the query. The query is executed in a single transaction using the snapshot isolation level, and the column specified in the ORDER BY clause ensures column uniqueness.

```
-- Ensure the database can support the snapshot isolation level set for the
query.
IF (SELECT snapshot_isolation_state FROM sys.databases WHERE name =
N'AdventureWorks2008R2') = 0
    ALTER DATABASE AdventureWorks2008R2 SET ALLOW_SNAPSHOT_ISOLATION ON;
GO

-- Set the transaction isolation level  to SNAPSHOT for this query.
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
GO

-- Beging the transaction
BEGIN TRANSACTION;
GO
-- Declare and set the variables for the OFFSET and FETCH values.
DECLARE @StartingRowNumber int = 1
      , @RowCountPerPage int = 3;

-- Create the condition to stop the transaction after all rows have been
returned.
WHILE (SELECT COUNT(*) FROM HumanResources.Department) >= @StartingRowNumber
BEGIN

-- Run the query until the stop condition is met.
SELECT DepartmentID, Name, GroupName
FROM HumanResources.Department
ORDER BY DepartmentID ASC
    OFFSET @StartingRowNumber - 1 ROWS
    FETCH NEXT @RowCountPerPage ROWS ONLY;

-- Increment @StartingRowNumber value.
SET @StartingRowNumber = @StartingRowNumber + @RowCountPerPage;
CONTINUE
END;
GO
COMMIT TRANSACTION;
GO
```

10 Contained Databases

10.1 Problems with non contained databases

10.1.1 When a database is moved from one to another SQL Server instance, instance specific information such as logins and agent job cannot be moved. These tasks need to be recreated again.

10.1.2 Application deployment in a server from the development server may cause issues due to environmental mismatch.

10.1.3 Login and job information are available at instance level. So, users need to be granted permission at the instance level causing security issue.

10.1.4 It is difficult to maintain and administer a single database independently.

10.2 Introduction to Contained database

10.2.1 This is a self-contained database i.e. it includes all the database settings and required metadata information.

10.2.2 This is instance or server independent. Users can be authenticated in the database itself.

10.2.3 This database can be moved easily to another instance or server as there is no external dependency.

10.2.4 Tempdb collation issues are resolved with contained database. Temporary objects are created as per the collation setting of the database, not of tempdb.

10.3 Contained database terms:

10.3.1 Application boundary

This is the boundary between server instance and the application code

10.3.2 Application model

This is the place inside the application boundary where the applications are developed and managed

10.3.3 Contained

User entity which resides entirely within the application boundary

10.3.4 Uncontained

User entity which crosses the application boundary

10.3.5 Non-contained database

A database with containment set to NONE

10.3.6 Fully contained database

A fully contained database does not allow any objects or functions that cross the application boundary. Fully contained databases are not currently available

10.3.7 Partially contained database

Partially contained database is a contained database that allows features that cross the application boundary.

10.3.8 Contained user

Two types of users are there:

10.3.8.1 Contained database users with password which are authenticated by the database

10.3.8.2 Windows principals that can directly connect to the database. Master database logins are not required for these users.

10.4 Contained database creation steps

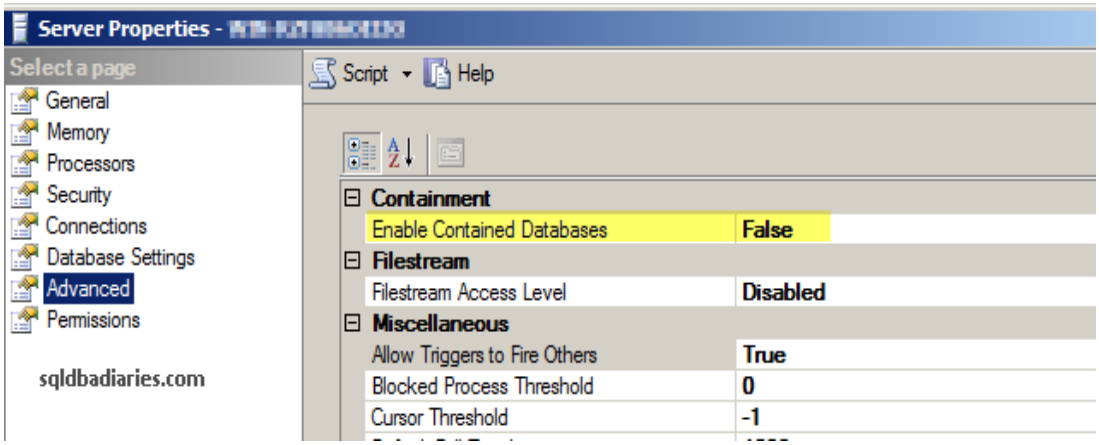
10.4.1 Enable contained database

10.4.1.1 SQL Code

```
--Enabled Advanced options

sp_configure 'show advanced', 1;

RECONFIGURE WITH OVERRIDE;

go

--Enabled Database Containment

sp_configure 'contained database authentication', 1;

RECONFIGURE WITH OVERRIDE;

Go
```

10.4.1.2 Through SSMS

It can be enabled by setting the Enable Contained Databases to TRUE using SSMS GUI.



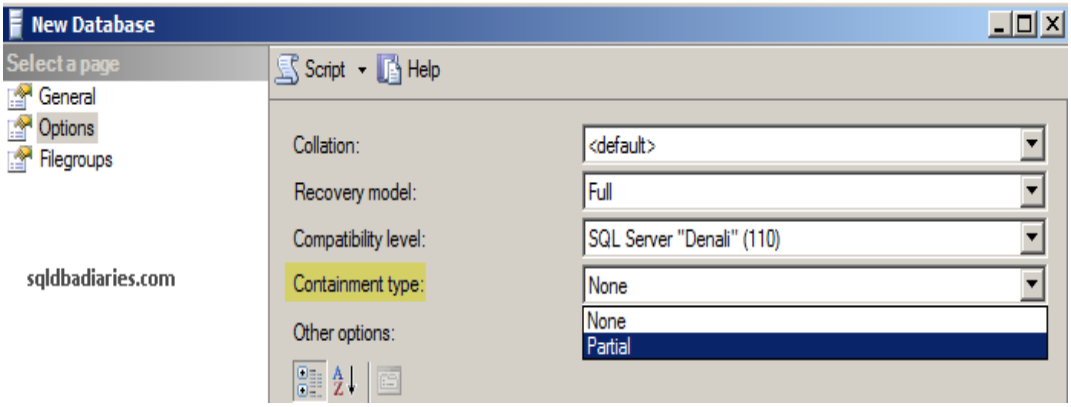
10.4.2 Create/Alter contained database

After enabling contained databases, the next step is to create a Contained Database. While creating the database, changing the Containment type to Partial will make the database as Contained. By default this option is set to None.

10.4.2.1 SQL Code

```
CREATE DATABASE [TestContainedDB]
CONTAINMENT = PARTIAL
ON  PRIMARY
( NAME = N'TestContainedDB', FILENAME = N'C:\Program
Files\Microsoft SQL
Server\MSSQL11\MSSQL\DATA\TestContainedDB.mdf' , SIZE = 4096KB
, MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
( NAME = N'TestContainedDB_log', FILENAME = N'C:\Program
Files\Microsoft SQL
Server\MSSQL11\MSSQL\DATA\TestContainedDB_log.ldf' , SIZE =
1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO


USE [master]
GO
ALTER DATABASE [databasename] SET CONTAINMENT = PARTIAL

GO
```

10.4.2.2 Through SSMS

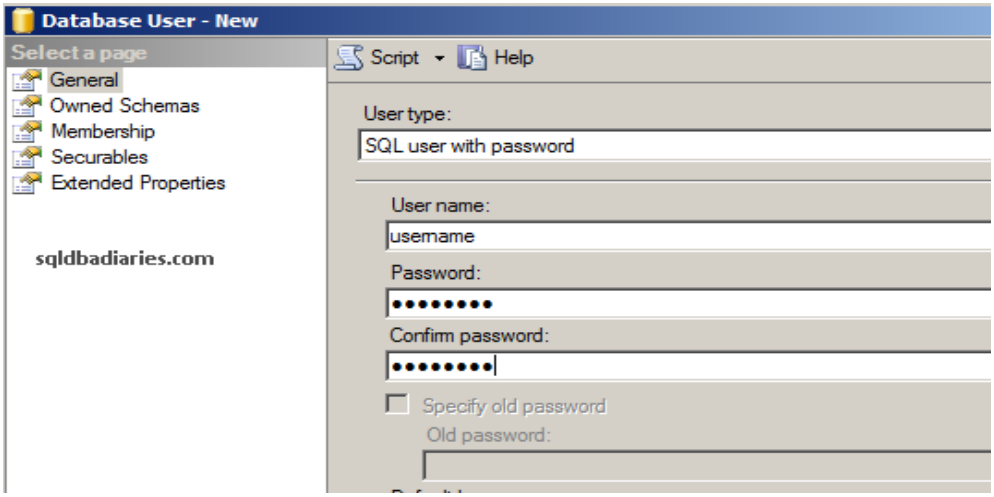


10.4.3 Create user in contained database

10.4.3.1 SQL Code

```
--SQL User

USE [ContainedDB1];
GO
CREATE USER [MyUser] WITH PASSWORD = 'DB1';
GO

USE [ContainedDB2];
GO
CREATE USER [MyUser] WITH PASSWORD = 'DB2';
GO



--Windows User

USE [ContainedDB1];
GO
CREATE USER [Domain\User]; -- note user@domain is not supported
GO
```

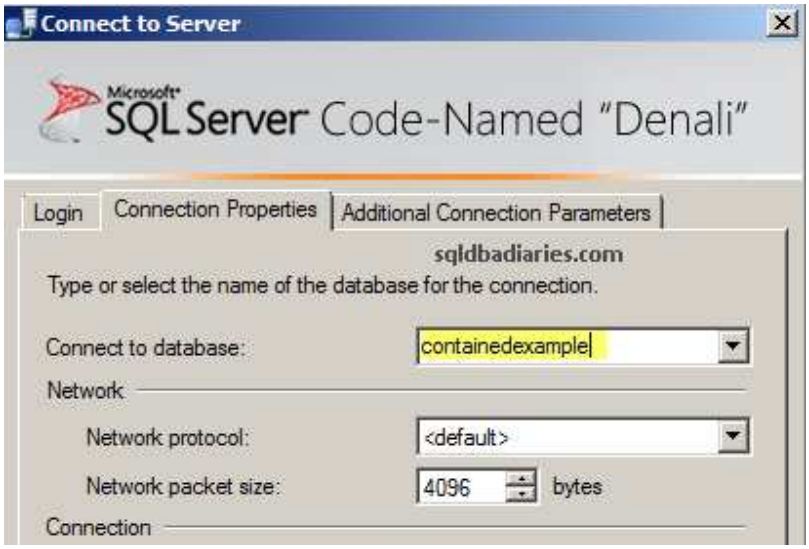
10.4.3.2 Through SSMS



10.4.4 Login to the contained database

In SSMS just entering the user name and password of this user is not enough to connect to the instance. Since this user does not have an associated login, SSMS raises the following error.
Error: 18456, Severity: 14, State: 5.
Login failed for user 'containeduser'. Reason: Could not find a login matching the name provided.

In order to connect using the credentials of a *user with password* through SSMS, the *Contained Database* name needs to be entered in the *Connect to Database* drop down box.



10.4.5 Uncontaining the database

```
USE [master]
GO
ALTER DATABASE [databasename] SET CONTAINMENT = PARTIAL
GO
```

The user with password within the contained database should be dropped first to change the *Containment* option to *None.*

10.5 Limitations

1.1.1. A database cannot be contained if is part of replication, change data capture and change tracking.

1.1.2. Fully contained database can't be created.

11 Column-store indexes (CSI)

11.1 Introduction to column-stores

Column-stores were envisioned for analytics in data warehouse loads which had de-normalized start schema where fact tables have many columns and large number of rows. However, for most ad hoc analytic queries, only a few columns of the table are used, which made sense to store columns separately rather than together as a row.

A column store, in the scope of Relational Database Management Systems (RDBMS), differs from the traditional products like Oracle or Microsoft SQL Server, in the way it stores data physically. Let's take a table for example –

ID	First name	Middle name	Last name	Area of interest
1	Joshua	S	Smith	Databases
2	Steve	N	Jones	Front-end
3	John	B	Doe	Networks
4	Paul	S	Silver	Artificial Intelligence

A traditional row-store would have stored the contents of the above table in a physical page as shown below (not an exact representation) –

RS-Page-1

1,Joshua,S,Smith,Databases; 2,Steve,N,Jones,Front-end; 3,John,B,Doe,Networks; 4,Paul,S,Silver,Artifical Intelligence

But a column-store would store the table in 5 physical pages, one for each of the columns as shown below (again, not an exact representation) –

CS-Page-1

1,2,3,4

CS-Page-2

Joshua,Steve,John,Paul

CS-Page-3

S,N,B,S

CS-Page-4

Smith,Jones,Doe,Silver

CS-Page-5

Databases,Front-end,Networks,Artifical Intelligence

11.2 Column-store advantages

There are advantages to this approach from the perspective of Analytics (OLAP) where the workload is mostly SELECT queries on top of huge amount of data.

It needs to be stressed here that this architecture is not suitable for Transactional processing (OLTP) requirements where data modification (INSERT/UPDATE/DELETE) queries happen often, and where SELECT queries often deal with a fraction of the rows in a table.

11.2.1 Less data during query processing

In OLAP scenarios where we deal with de-normalized schemas, it is not unusual to have tables with many columns, but for a given query only a fraction of the number of columns is used. In our example in Section 2, for getting the distinct first names, we need to load only CS-Page-2 in a column-store which has about 30 characters rather than the entire RS-Page-1 which has more than 100 characters (just 1/3rd of the data needs to be taken from disk) – consider the savings that we'll have in a billion-row table common in analytics scenarios.

One might argue that this can be achieved by creating an index in a row-store. But there is an overhead of having another data structure in addition to the actual table, which is not the case with column stores.

It is to be noted here that more the number of columns in a table is used for a query, the advantages of using a column-store decline proportionally.

11.2.2 Efficient compression

Compression is a methodology in which recurring patterns of data is replaced with some kind of pointer, much smaller in size. As is evident from the example in Section 2, since each column will have same type (homogenous) of data, compression in column-store is far superior compared to a row-store with differing types (heterogenous) of data.

Scenarios are common in which a table stored in a column-store occupies much lesser footprint than when it is stored as a raw file. Contrast this with row-store where the table size in DB is bloated because of additional page level information and indexes.

11.3 Column-store index (CSI) in SQL Server

While SQL Server is not converting its storage to be a column-store, Denali is being endowed with a column-store index, by which we mean that there is a row-store table on top of which a column-store index can be built. How much these two different physical storage architectures can reside side-by-side is yet to be seen.

11.3.1 Batch mode execution

As per the literature available (listed in References section), when a CSI is used, it is possible that the query optimizer will use a batch-mode execution rather than the regular row-mode execution.

11.3.2 Points to be considered before using CSI

Though there are definite benefits with a column-store for its target queries, there are also some limitations that are present with this feature in its current state.

11.3.2.1 *Separate data structure*

CSI will always be a separate data structure, thereby occupying its own space. There has to be an underlying table with a row-store storage above which CSI will need to be created.

This is probably to preserve the original row-store physical storage architecture of SQL Server and to allow the optimizer to choose this access method should an appropriate query (e.g. seek) be fired.

11.3.2.2 *Only one CSI per table*

Although listed as a limitation, it is unnecessary to have more than one CSI. If we include all the columns in a table (those columns that have the allowed data types), we're covered.

11.3.2.3 *No DML operations*

Once a column-store index is created on a table, no DML operations (INSERT/UPDATE/DELETE) are allowed in it. To do so, the CSI needs to be dropped. Once the table is manipulated, the CSI can be created again.

Since any data warehouse environment will have only batch loads at large intervals (something like once-a-day), this may not be an issue. But for those environments where frequent updates are required, this poses a significant limitation.

11.3.2.4 *Not all data types are supported*

The data types that are not supported as per this link are –

- decimal or numeric with precision > 18
- datetimeoffset with precision > 2
- binary
- varbinary

- image
- text
- ntext
- varchar(max)
- nvarchar(max)
- cursor
- hierarchyid
- timestamp
- uniqueidentifier
- sqlvariant
- xml

11.3.3 Metadata

Metadata about CSI is stored in 3 system tables -

- sys.column_store_index_stats
- sys.column_store_segments
- sys.column_store_dictionaries

12 Enhanced Collation Support

SQL Server provides data types such as nchar and nvarchar to store Unicode data. These data types encode text in a format called *UTF-16*. The Unicode Consortium allocates each character a unique codepoint, which is a value in the range 0x0000 to 0xFFFF. The most frequently used characters have codepoint values that will fit into a 16-bit word in memory and on disk, but characters with codepoint values larger than 0xFFFF require two consecutive 16-bit words. These characters are called *supplementary characters*, and the two consecutive 16-bit words are called *surrogate pairs*.

12.1 Supplementary Characters

- Supplementary characters can be used in ordering and comparison operations in collation versions 90 or greater.
- All _100 level collations support linguistic sorting with supplementary characters.
- Supplementary characters are not supported for use in metadata, such as in names of database objects.
- Introduced in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), a new family of supplementary character (SC) collations can be used with the data types nchar, nvarchar and sql_variant. For example: Latin1_General_100_CI_AS_SC, or if using a Japanese collation, Japanese_Bushu_Kakusu_100_CI_AS_SC.

  The SC flag can be applied to:
  - Version 90 Windows collations
  - Version 100 Windows collations

  The SC flag cannot be applied to:
  - Version 80 non-versioned Windows collations
  - The BIN or BIN2 binary collations
  - The SQL* collations

12.2 Limitations of Supplementary Characters

The limitations listed at the bottom of the topic apply only to supplementary characters when a SC collation is NOT specified. The limitations do not apply when a SC collation is used.

String Function or Operator	With an SC Collation	Without an SC Collation
CHARINDEX LEN PATINDEX	The UTF-16 surrogate pair is counted as a single codepoint.	The UTF-16 surrogate pair is counted as two codepoints.

LEFT REPLACE REVERSE RIGHT SUBSTRING STUFF	These functions treat each surrogate pair as a single codepoint and work as expected.	These functions may split any surrogate pairs and lead to unexpected results.
NCHAR	Returns the character corresponding to the specified Unicode codepoint value in the range 0 to 0x10FFFF. If the value specified lies in the range 0 through 0xFFFF, one character is returned. For higher values, the corresponding surrogate is returned.	A value higher than 0xFFFF returns NULL instead of the corresponding surrogate.
UNICODE	Returns a UTF-16 codepoint in the range 0 through 0x10FFFF.	Returns a UCS-2 codepoint in the range 0 through 0xFFFF.
Match One Character Wildcard Wildcard - Character(s) Not to Match	Supplementary characters are supported for all wildcard operations.	Supplementary characters are not supported for these wildcard operations. Other wildcard operators are supported.

13 Database Engine Tuning Advisor Enhancements

In Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3), you can use the query plan cache as a Database Engine Tuning Advisor (DTA) workload. By doing this, you can avoid having to manually create a workload from a script or trace file. When you specify the plan cache as the DTA workload, the Database Engine Tuning Advisor selects the top 1,000 events to use for analysis. The number of events can be changed using the **–n** option of the DTA utility

13.1 To tune a database by using the plan cache

1. Launch Database Engine Tuning Advisor, and log into an instance of SQL Server. For more information, see Start Database Engine Tuning Advisor.
2. On the General tab, type a name in Session name to create a new tuning session.
3. Select Plan Cache as the workload option. Database Engine Tuning Advisor selects the top 1,000 events from the plan cache to use for analysis.
4. Select the database or databases that you want to tune, and optionally from Selected Tables, choose one or more tables from each database. To include cache entries for all databases, from Tuning Options, click Advanced Options and then check Include plan cache events from all databases.
5. Check Save tuning log to save a copy of the tuning log. Clear the check box if you do not want to save a copy of the tuning log.You can view the tuning log after analysis by opening the session and selecting the Progress tab.
6. Click the Tuning Options tab and select from the options listed there.
7. Click Start Analysis.

14 Installation Enhancements

SQL Server Code-Named “Denali” introduces the following changes to SQL Server Setup:

14.1 Installing Prerequisites during SQL Server Code-Named “Denali” Setup

- Windows PowerShell 2.0 is a prerequisite for installing SQL Server Database Engine components and SQL Server Management Studio, but Windows PowerShell is no longer installed by SQL Server Setup. If Windows PowerShell 2.0 is not present on your computer, you can enable it by following the instructions on the Windows Management Framework page.
- .NET Framework 4 is a prerequisite for installing Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3). SQL Server Setup installs the .NET Framework 4 during the feature installation step. Setup does not install the .NET Framework 4 when installing SQL Server Express on a computer that is running Windows Server 2008 R2 SP1 Server Core operating system. For more information about installing SQL Server Express on Server Core, see Installing SQL Server "Denali" on Server Core.
- The .NET Framework 3.5 SP1 is a requirement for Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) when you select Database Engine, Replication, Master Data Services, Reporting Services, Data Quality Services (DQS), or SQL Server Management Studio, and it is no longer installed by SQL Server Setup.
- If you run Setup on a computer with the Windows Vista SP2 or Windows Server 2008 SP2 operating system, and you do not have the .NET Framework 3.5 SP1 installed, SQL Server Setup requires you to download and install the .NET Framework 3.5 SP1 before you can continue with the SQL Server installation.
- If you run Setup on a computer with the Windows 7 SP1 or Windows Server 2008 R2 SP1 operating system, you must enable the .NET Framework 3.5 SP1 before you install Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).
- To make sure that the Visual Studio component can be installed correctly, SQL Server requires you to install an update. SQL Server Setup checks for the presence of this update and then requires you to download and install the update before you can continue with the SQL Server installation. To avoid the interruption during SQL Server Setup, you can download and install the update before running SQL Server Setup as described below (or install all the updates for .NET 3.5 SP1 available on Windows Update):
- If you install Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) on a computer with the Windows Vista SP2 or Widows Server 2008 SP2 operating system, you can get the required update from http://support.microsoft.com/?kbid=956250
- If you install Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) on a computer with the Windows 7 SP1 or Windows Server 2008 R2 SP1 operating system, this update is already installed on the computer.

14.2 Changes to Operating System Requirements

Starting with Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), Service Pack 1 is the minimum requirement for Windows 7 and Windows Server 2008 R2 operating systems.

14.3 Data Quality Services

You can now install Data Quality Services (DQS) using the SQL Server Setup. For more information, see Installing Data Quality Services.

14.4 Product Update

Product Update is a new feature in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) Setup. It integrates the latest product updates with the main product installation so that the main product and its applicable updates are installed at the same time. For more information, see Product Updates in SQL Server "Denali" Installation

14.5 Server Core Installation

Starting with Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), we can install SQL Server on Windows Server 2008 R2 Server Core SP1. For more information, see Installing SQL Server "Denali" on Server Core

14.6 SQL Server multi-subnet clustering

You can now configure a SQL Server failover cluster using clustered nodes on different subnets. For more information, see SQL Server Mutli-Subnet Failover Cluster Configuration.

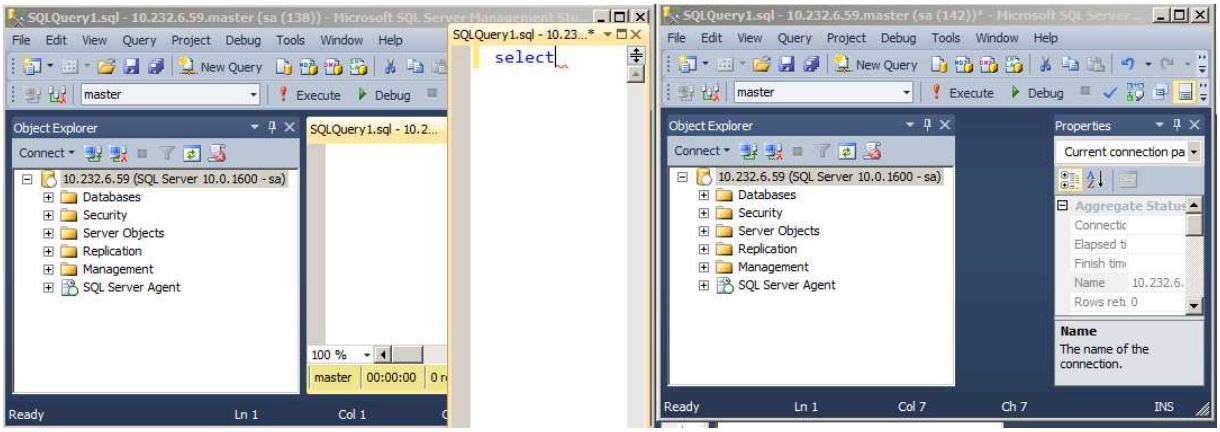
14.7 SMB file share is a supported storage option

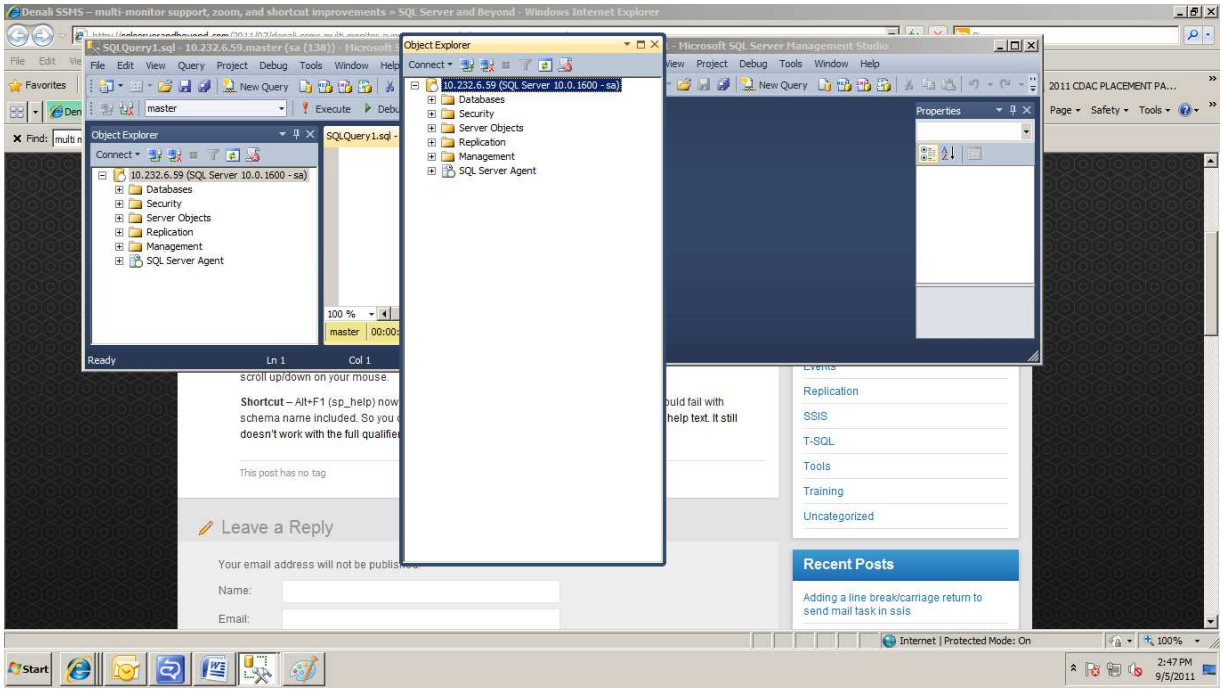
System databases (Master, Model, MSDB, and TempDB), and Database Engine user databases can be installed on a file share on an SMB file server. This applies to both SQL Server stand-alone and SQL Server failover cluster installations. For more information, see Database Engine Configuration - Data Directories.

- Local Disk is now a supported storage option for tempdb for SQL Server failover cluster installations. For more information on recommendations when using a local disk in SQL Server failover cluster installations, see Database Engine Configuration - Data Directories.
- BUILTIN\administrators and Local System (NT AUTHORITY\SYSTEM) are not automatically provisioned in the sysadmin fixed server role.
- The protection of operating services under a per-service SID is now extended to all operating systems. For more information, see Configure Windows Service Accounts and Permissions.
- Setup now offers default accounts for the SQL Server services whenever possible.
- The Active Directory Helper service is no longer installed because it is no longer needed.
- Itanium support: SQL Server Itanium editions are no longer supported.

15 Multi Monitor Support

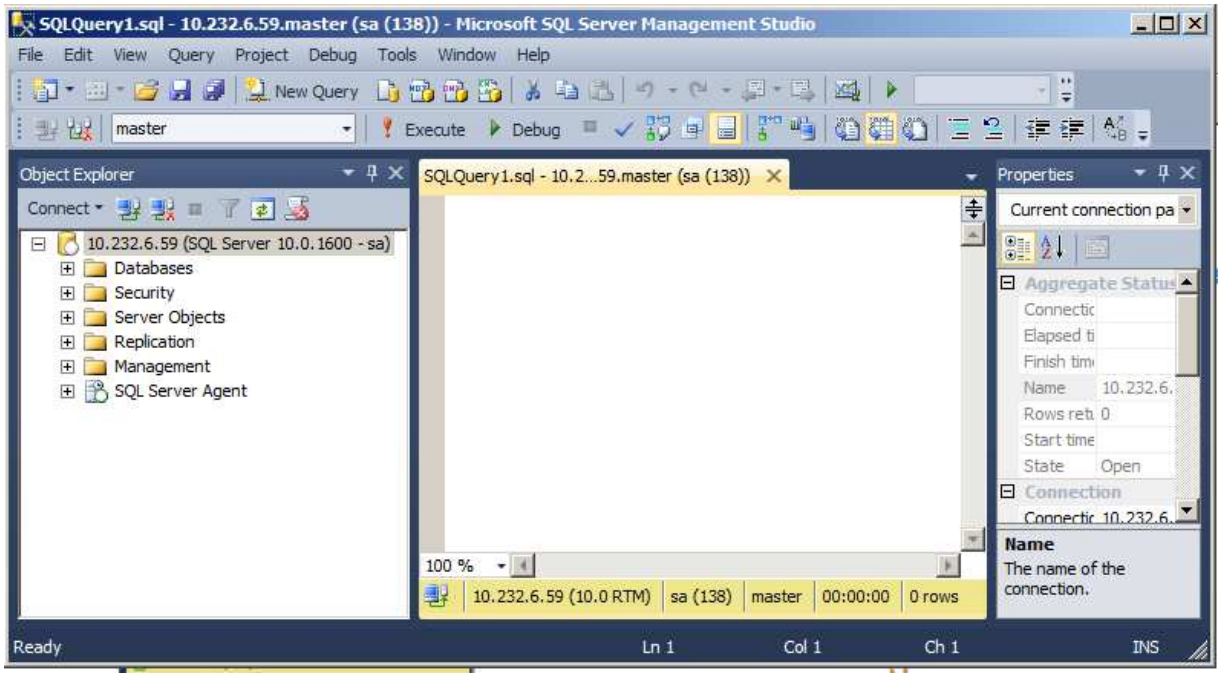
With the introduction of Multi-Monitor support in Sql Server Denali CTP3, there is no need to open two SSMS to split work out on each instance, we can now drag a query window, object explorer etc over to other instance(s).





16 Zoom

There is a dropdown given in bottom- left corner of the window to zoom the query editor windows to the size needed.



17 Replication Enhancement

Denali CTP3 of SQL Server introduces several new features and improvements to replication.

- Replication supports 15,000 partitions
- AlwaysOn Publisher Failover Support

17.1 Replication Backward Compatibility

Topics in the backward compatibility section describe changes in behavior between versions of Microsoft SQL Server replication. It is important to understand backward compatibility to upgrade or if there is more than one version of SQL Server in a replication topology.

17.1.1 Deprecated Features in SQL Server Replication

Replication features that have been retained in Microsoft Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) for backward compatibility, but which will be removed in a future version of SQL Server.

Refer: http://msdn.microsoft.com/en-us/library/ms143550(v=SQL.110).aspx

17.1.2 Behavior Changes in SQL Server Replication

Replication features that have changed in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).

Refer: http://msdn.microsoft.com/en-us/library/ms143733 (v=SQL.110).aspx

17.1.3 Breaking Changes in SQL Server Replication

Replication feature changes that might require changes to applications.

Refer: http://msdn.microsoft.com/en-us/library/ms143470(v=SQL.110).aspx

http://msdn.microsoft.com/en-us/library/bb500342(v=sql.110).aspx

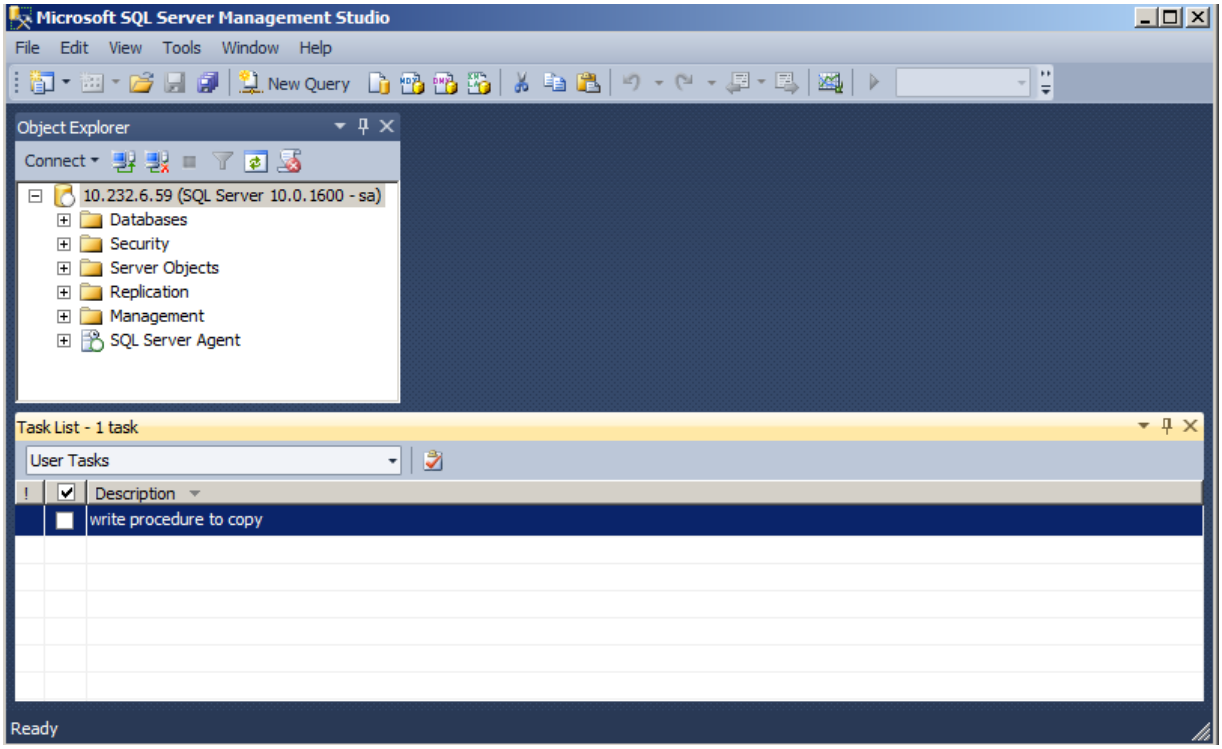
18 Task List

Sometime we need external tools to list our day to day tasks mostly when work for the SQL Server as it does not have any built in features with previous version. We can follow the priority and sequence of the tasks from the list and also can have idea for the current or future tasks from list with the tool.

SQL Server 2011 came up with Task List as a solution to above problem.

Implementation:

Task list can be accessed from the **View --> Task List or Ctrl+\T**. It will appear as following. We can add new task in list, set the priority and complete the tasks in Task List. Here is image which demonstrates all the things.



19 SSIS Enhancements

19.1 Troubleshooting Performance and Data Issues

This release provides additional views, stored procedures, and stored functions to help you troubleshoot performance and data issues.

19.2 Troubleshooting Capability

19.2.1 Get performance statistics and other information for an execution

Related Views, Procedures, and Functions

19.2.1.1 *catalog.executions (SSISDB Database)*

Displays the instances of package execution in the Integration Services catalog. Packages that are executed with the Execute Package task run in the same instance of execution as the parent package.

Refer: http://msdn.microsoft.com/en-us/library/ff878089(v=SQL.110).aspx

19.2.1.2 *catalog.execution_component_phases*

Displays the time spent by a data flow component in each execution phase.

Refer: http://msdn.microsoft.com/en-us/library/hh230981(v=SQL.110).aspx

19.2.1.3 *dm_execution_performance_counters (SSISDB Database)*

Returns the performance statistics for the packages in an execution.

Syntax

dm_execution_performance_counters [@execution_id =] execution_id

Argument
[@execution_id =] *execution_id*

The unique identifier of the execution that contains one or more packages. If an execution ID is not specified, performance statistics for executions are returned. If you are a member of ssis_admin database role, performance statistics for all running executions are returned. If you are not a member of the ssis_admin database role, performance statistics for the running executions for which you have read permissions, are returned. The *execution_id* is a BigInt.

Refer: http://msdn.microsoft.com/en-us/library/hh230983(v=SQL.110).aspx

19.2.2 Add, remove, and query data taps in a package data flow Related Views, Procedures, and Functions

19.2.2.1 *catalog.add_data_tap*

Adds a data tap on the output of a component in a package data flow.

Syntax

add_data_tap [@execution_id =] execution_id
[@task_package_path =] task_package_path
[@dataflow_path_id_string =] dataflow_path_id_string
[@data_filename =] data_filename
[@num_rows =] num_rows

Arguments

[@execution_id =] *execution_id*

The execution ID for the execution that contains the package. The *execution_id* is a bigint.

[@task_package_path =] *task_package_path*

The package path for the data flow task. The path is case-sensitive. The task_package_path is a nvarchar(max).

[@dataflow_path_id_string =] *dataflow_path_id_string*

The identification string for the data flow path. In BI Development Studio, the **IdentificationString** property for the Data Flow path specifies the string. The *dataflow_path_id_string* is a nvarchar(4000).

[@data_filename =] *data_filename*

The name of the file that stores the tapped data. If the data flow task executes inside a Foreach Loop or a For Loop container, separate files store tapped data for each iteration of the loop. Each file is prefixed with a number that corresponds to an iteration. The *data_filename* is a nvarchar(4000).

[@nums_rows =] *nums_rows*

The number of rows that are captured during the data tap. If this value is not specified, all rows are captured. The *nums_rows* is an int.

Refer: http://msdn.microsoft.com/en-us/library/hh230989(v=SQL.110).aspx

19.2.2.2 *catalog.remove_data_tap*

Removes a data tap from a component output that is in an execution.

Syntax

```
remove_data_tap [@data_tap_id =] data_tap_id
```

Arguments

[@data_tap_id =] *data_tap_id*

The unique identifier for the data tap that is created by using the catalog.add_data_tap stored procedure. The *data_tap_id* is bigint.

Refer: http://msdn.microsoft.com/en-us/library/hh230990(v=SQL.110).aspx

19.2.2.3 *catalog.execution_data_taps*

Displays information for each data tap defined in an execution.

Refer: http://msdn.microsoft.com/en-us/library/hh230985(v=SQL.110).aspx

19.2.2.4 *catalog.execution_data_statistics*

This view displays a row each time a data flow component sends data to a downstream component. The information in this view can be used to compute the data throughput for a component.

Refer: http://msdn.microsoft.com/en-us/library/hh230986(v=SQL.110).aspx

19.2.3 Create a dump for a running package

19.2.3.1 *catalog.create_execution_dump*

Causes a running package to pause and create a dump file.

Syntax

```
create_execution_dump [@execution_id =] execution_id
```

Arguments

[@execution_id =] *execution_id*

The execution ID for the running package. The *execution_id* is bigint.

Refer: http://msdn.microsoft.com/en-us/library/hh230987(v=SQL.110).aspx

Set a parameter value in an instance of an execution

19.2.3.2 *catalog.set_execution_parameter_value (SSISDB Database)*

Sets the value of a parameter for an instance of execution in the Integration Services catalog.

Syntax

set_execution_parameter_value [@execution_id = execution_id

, [@object_type =] object_type

, [@parameter_name =] parameter_name

, [@parameter_value =] parameter_value

20 Security Enhancements

Security enhancements in the SQL Server Database Engine include provisioning during setup, new SEARCH PROPERTY LIST permissions, new user-defined server roles, and new ways of managing server and database roles.

20.1 Provisioning During Setup

To enhance role separation, BUILTIN\administrators and Local System (NT AUTHORITY\SYSTEM) are not automatically provisioned in the sysadmin fixed server role. Local administrators can still access the Database Engine when in single user mode.

SQL Server now supports Managed Service Accounts and Virtual Accounts when installed on Windows 7 or Windows Server 2008 R2. The protection of operating services under a per-service SID is now extended to all operating systems

20.2 New Permissions

New GRANT, REVOKE, and DENY permissions to a SEARCH PROPERTY LIST are available.

New GRANT, REVOKE, and DENY permissions to CREATE SERVER ROLE and ALTER ANY SERVER ROLE.

20.3 New Role Management

- User-defined server roles are now available. To manage user-defined server roles and to add and remove members from all server roles, use CREATE SERVER ROLE, ALTER SERVER ROLE, and DROP SERVER ROLE.
- sp_addsrvrolemember and sp_dropsrvrolemember are deprecated. Use ALTER SERVER ROLE instead.
- ALTER ROLE is modified to add or remove members from roles.
- sp_addrolemember and sp_droprolemember are deprecated. Use ALTER ROLE instead.
- IS_ROLEMEMBER is added to check the membership of database roles.

20.4 ALTER ROLE (Transact-SQL)

Changes the name of a database role.

Syntax

ALTER ROLE role_name WITH NAME = new_name

Arguments

- *role_name* Is the name of the role to be changed.
- WITH NAME =*new_name* Specifies the new name of the role. This name must not already exist in the database.

20.5 IS_ROLEMEMBER (Transact-SQL)

Indicates whether a specified database principle is a member of the specified database role.

Syntax

IS_ROLEMEMBER (‘role’ [, 'database_principal'])

Arguments

- ' *role* ' Is the name of the database role that is being checked. *role* is sysname.
- ' *database_principal* ' Is the name of the database user, database role, or application role to check. *database_principal* is sysname, with a default of NULL. If no value is specified, the result is based on the current execution context. If the parameter contains the word NULL will return NULL.

20.6 Default Schema for Groups

You can now define a default schema for a Windows group. When an object is created by a Windows user and when a default schema is not specified, SQL Server no longer automatically creates a schema.

20.7 SQL Server Audit Enhancements

Support for server auditing is expanded to include all editions of SQL Server. Database audits are limited to Enterprise, Datacenter, Developer, and Evaluation editions.

SQL Server Audit is now more resilient to failures to write to the audit log. For example, if the target directory is on a remote share and the network goes down, SQL Server Audit will now be able to recover once the network connection is re-established. In addition, a new option has been introduced to fail an operation that would otherwise generate an audit event to be written to a failed audit target. For more information, see the **FAIL_OPERATION** option for the **ON_FAILURE** event in CREATE SERVER AUDIT.

Previously, Audit logs could have an indeterminate number of log files or else be rolled-over after a predefined number. A new option has been introduced to cap the number of audit files without rolling over, in order to allow customers to control the amount of audit information collected without losing audit records. For more information, see the **MAX_FILES** option in CREATE SERVER AUDIT.

When possible, the audit log provides additional Transact-SQL stack frame information. In many cases, auditors can now determine whether a query was issued through a stored procedure or directly by an application.

SQL Server audit specifications now support a user-defined audit group. Audited events can be written to the audit log by using the new sp_audit_write (Transact-SQL) procedure. User-defined audit events allow applications to write custom information to the audit log, such as the name of the application user who has connected in cases where a common login is used to connect to SQL Server.

New columns are added to sys.server_file_audits, sys.server_audits, and sys.fn_get_audit_file to track user-defined audit events.

SQL Server Audit now supports the ability to filter audit events before they are written to the audit log. For more information, see the **WHERE** clause in CREATE SERVER AUDIT and ALTER SERVER AUDIT.

New audit groups support the monitoring of contained database users.The new audit options have been added to the audit dialog boxes in Management Studio.

20.8 Database Engine Access is allowed Through Contained Databases

Access to contained databases is permitted through contained database users which do not require logins. SQL Server system administrators should understand how contained databases change the SQL Server security model.

20.9 Hashing Algorithms

The HASHBYTES function now supports the SHA2_256, and SHA2_512 algorithms.

20.10 Further Deprecation of RC4

The RC4 algorithm is only supported for backward compatibility. New material can only be encrypted using RC4 or RC4_128 when the database is in compatibility level 90 or 100. (Not recommended.) Use a newer algorithm such as one of the AES algorithms instead. In Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) material encrypted using RC4 or RC4_128 can be decrypted in any compatibility level.

20.11 Certificate Key Length

When creating certificates, the maximum length of private keys imported from an external source is expanded from 3,456 to 4,096 bits.

20.12 Service Master Key and Database Master Key Encryption changes from 3DES to AES

Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) uses the AES encryption algorithm to protect the service master key (SMK) and the database master key (DMK). AES is a newer encryption algorithm than 3DES used in earlier versions. After upgrading an instance of the Database Engine to Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) the SMK and DMK should be regenerated in order to upgrade the master keys to AES. For more information about regenerating the SMK, see ALTER SERVICE MASTER KEY (Transact-SQL) and ALTER MASTER KEY (Transact-SQL).

Refer: http://msdn.microsoft.com/en-us/library/ms187788(v=SQL.110).aspx

http://msdn.microsoft.com/en-us/library/ms186937(v=SQL.110).aspx

20.13 Certificates can be Created from Binary

CREATE CERTIFICATE (Transact-SQL) has the FROM BINARY option to allow specifying the bits of an ASN encoded certificate.

Refer: http://msdn.microsoft.com/en-us/library/ms187798(v=SQL.110).aspx

21 Availability Enhancements

21.1 Database Mirroring's Drawbacks

- SQL Server was able to read the mirror's copy of the data to accomplish page repairs, but there wasn't any ability to do something helpful with the data. There wasn't direct access the database. The only best thing which could be done was take a snapshot of that database and query the snapshot, but that snapshot was frozen in time, it wasn't useful to shed load from the production server.
- Unlike log shipping and replication, mirroring only allowed for two SQL Servers to be involved. We could either use mirroring for high availability inside the same datacenter, OR use it for disaster recovery with two servers in different datacenters, but not both. Due to this limitation, a common HA/DR scenario involved using a cluster for the production server (giving local high availability in the event of a server failure) combined with asynchronous mirroring to a remote site. This worked fairly well.
- The next problem: database failovers are database-level events. DBAs can fail over one database from the principal to the secondary server, but can't coordinate the failover of multiple databases simultaneously. In applications that required more than one database, this made automatic failover a non-option. We couldn't risk letting SQL Server fail over just one database individually without failing over the rest as a group. Even if we tried to manage this manually, database mirroring sometimes still ran into problems when more than ten databases on the same server were mirrored.
- Database mirroring didn't protect objects outside of the database, such as SQL logins and agent jobs. SQL Server 2008 R2 introduced contained databases (DACs), a packaged set of objects that included everything necessary to support a given database application.

21.2 High Availability Solutions

SQL Server high-availability solutions improve the availability of servers or databases. A high-availability solution masks the effects of a hardware or software failure and maintains the availability of applications so that the perceived downtime for users is minimized.

SQL Server provides several options for creating high availability for a server or database. High-availability options include the following:

- Failover clustering

  Failover clustering provides high-availability support for an entire instance of SQL Server. A failover cluster is a combination of one or more nodes, or servers, with two or more shared disks. Applications are each installed into a Windows Server Failover Cluster (WSFC) cluster group, known as a resource group. At any time, each resource group is owned by only one node in the cluster. The application service has a virtual name that is independent of the node names, and is referred to as the failover cluster instance name. An application can connect to the failover cluster instance by referencing the failover cluster instance name. The application does not have to know which node hosts the failover cluster instance.

  A SQL Server failover cluster instance appears on the network as a single computer, but has functionality that provides failover from one node to another if the current node becomes unavailable. For example, during a non-disk hardware failure, operating system failure, or planned operating system upgrade, you can configure an instance of SQL Server on one node of a failover cluster to fail over to any other node in the disk group.

  A failover cluster that has all the nodes on the same subnet does not protect against disk failure. You can use failover clustering to reduce system downtime and provide higher application availability. Failover clustering is supported in SQL Server Datacenter, SQL Server Enterprise and SQL Server Developer, SQL Server Evaluation and, with some restrictions, in SQL Server Standard.

- Multi-subnet Failover Clustering

  A multi-subnet failover cluster typically has multiple storage arrays. This ensures that in the event of a failure on one storage array, there are other copies of data available. Because of this, multi-subnet failover clustering provides a disaster recovery solution in addition to high availability.

- AlwaysOn Availability Groups

AlwaysOn Availability Groups is an enterprise-level high-availability and disaster recovery solution introduced in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) to enable you to maximize availability for one or more user databases. AlwaysOn Availability Groups requires that the SQL Server instances reside on Windows Server Failover Clustering (WSFC) nodes.

- Database mirroring

Database mirroring is primarily a software solution to increase database availability by supporting almost instantaneous failover. Database mirroring can be used to maintain a single standby database, or *mirror database*, for a corresponding production database that is referred to as the *principal database*.

The mirror database is created by restoring a database backup of the principal database with no recovery. This makes the mirror database is inaccessible to clients. However, you can use it indirectly for reporting by creating a database snapshot on the mirror database. The database snapshot provides clients with read-only access to the data in the database as it existed when the snapshot was created.

Each database mirroring configuration involves a *principal server* that contains the principal database, and a mirror server that contains the mirror database. The mirror server continuously brings the mirror database up to date with the principal database.

Database mirroring runs with either synchronous operation in high-safety mode, or asynchronous operation in high-performance mode. In high-performance mode, the transactions commit without waiting for the mirror server to write the log to disk, which maximizes performance. In high-safety mode, a committed transaction is committed on both partners, but at the risk of increased transaction latency.

In its simplest configuration, database mirroring involves only the principal and mirror servers. In this configuration, if the principal server is lost, the mirror server can be used as a warm standby server, with possible data loss. High-safety mode supports an alternative configuration, high-safety mode with automatic failover. This configuration involves a third server instance, known as a *witness*, which enables the mirror server to act as a hot standby server. Failover from the principal database to the mirror database typically takes several seconds.

Since SQL Server 2005 Service Pack 1 (SP1), database mirroring partners and witnesses have been supported by Standard Edition and Enterprise Edition. But the partners must use the same

edition, and asynchronous database mirroring (high-performance mode) is supported only by Enterprise Edition. Witnesses are also supported by Workgroup Edition and Express Edition.

- Log shipping

  Like database mirroring, log shipping operates at the database level. You can use log shipping to maintain one or more warm standby databases for a corresponding production database that is referred to as the *primary database*. Standby databases are also referred to as *secondary databases*. Each secondary database is created by restoring a database backup of the primary database with no recovery, or with standby. Restoring with standby lets you use the resulting secondary database for limited reporting.

  A log shipping configuration includes a single primary server that contains the primary database, one or more secondary servers that each have a secondary database, and a monitor server. Each secondary server updates its secondary database at set intervals from log backups of the primary database. Log shipping involves a user-modifiable delay between when the primary server creates a log backup of the primary database and when the secondary server restores the log backup. Before a failover can occur, a secondary database must be brought fully up-to-date by manually applying any unrestored log backups.

  Log shipping provides the flexibility of supporting multiple standby databases. If you require multiple standby databases, you can use log shipping alone or as a supplement to database mirroring. When these solutions are used together, the current principal database of the database mirroring configuration is also the current primary database of the log shipping configuration.

- Replication

  Replication uses a publish-subscribe model. This lets a primary server, referred to as the Publisher, distribute data to one or more secondary servers, or Subscribers. Replication enables real-time availability and scalability across these servers. It supports filtering to provide a subset of data at Subscribers, and also allows for partitioned updates. Subscribers are online and available for reporting or other functions, without query recovery. SQL Server offers three types of replication: snapshot, transactional, and merge. Transactional replication provides the lowest latency and is usually used for high availability.

  Replication is supported in all editions of SQL Server. Replication publishing is not available with SQL Server Express or SQL Server Compact.

- A well designed and implemented backup and restore strategy is **important** to any high-availability solution

- Scalable shared databases

  The scalable shared database feature lets you scale out a read-only database built exclusively for reporting. The reporting database must reside on a set of dedicated, read-only volumes whose primary purpose is hosting the database. By using commodity hardware for servers and volumes, you can scale out a reporting database that provides the same view of the reporting data on multiple reporting servers. This feature also allows a smooth update path for the reporting database.

21.3 AlwaysOn SQL Server Failover Cluster Instances

- **Multi-subnet failover clusters:** A SQL Server multi-subnet failover cluster is a configuration where each failover cluster node is connected to a different subnet or different set of subnets. These subnets can be in the same location or in geographically dispersed sites. Clustering across geographically dispersed sites is sometimes referred to as Stretch clusters. As there is no shared storage that all the nodes can access, data should be replicated between the data storage on the multiple subnets. With data replication, there is more than one copy of the data available. Therefore, a multi-subnet failover cluster provides a disaster recovery solution in addition to high availability.
- **Flexible failover policy for cluster health detection:** In a SQL Server failover cluster instance, only one node can own the cluster resource group at a given time. The client requests are served through this node for that failover cluster instance. In the case of a failure, the group ownership is moved to another node in the failover cluster. This process is called failover. The improved failure detection introduced in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), and addition of failure condition level property allows you to configure a more flexible failover policy.

21.4 AlwaysOn Availability Groups

Deploying AlwaysOn Availability Groups involves creating and configuring one or more availability groups. An availability group is a container that defines a set user databases (availability databases) to fail over as a single unit, and a set of *availability replicas* to host copies of each availability database. Each availability group requires at least two availability replicas: the primary availability replica and one secondary availability replica.

AlwaysOn Availability Groups provides a rich set of options that improve database availability and that enable improved resource use. The key components are as follows:

- Multiple secondary replicas: one primary replica and up to four secondary replicas.

- Alternative availability modes: *Asynchronous-commit mode* and *Synchronous-commit mode*.
- Several failover modes: *automatic failover*, *planned manual failover*, and *forced manual failover*.
- Active secondary replicas, as follows:
    - Read-only access to the secondary replicas.
    - Performing backup operations on secondary replicas.

  Active secondary capabilities improve IT efficiency and reduce cost through better resource utilization of secondary hardware. In addition, offloading read-intent applications and backup jobs to secondary replicas helps to improve performance on the primary replica.

- Availability group listeners that provide fast application failover after an availability group fails over.
- A flexible failover policy for each availability group to provide some control over the automatic failover process.
- Automatic page repair for protection against page corruption.
- Forcing WSFC quorum (*forced quorum*). Encryption and compression, which provide a secure, high performing transport.
- Interoperation with the following SQL Server features:
    - Change data capture
    - Change tracking
    - Contained databases
    - FILESTREAM
    - FileTable
    - Remote Blob Store (RBS)
    - Service Broker
    - SQL Server Agent

21.5 Indirect Checkpoints

The indirect checkpoints feature provides a database-specific alternative to automatic checkpoints, which are configured by a server property. Indirect checkpoints implement a new checkpointing algorithm for the Database Engine. This algorithm provides a more accurate guarantee of database recovery time in the event of a crash or a failover than is provided by automatic checkpoints. To ensure that database recovery does not exceed allowable downtime for a given database, you can specify the maximum allowable downtime for that database.

21.6 How to Implement HADR (a high-availability and disaster recovery solution)

Step 1: Enable the HADR service on both clustered DENALI instances

- Open SQL Server Configuration Manager
- Select SQL Server Services
- Right-click on your SQL Server and select properties
- Select the SQL HADR tab and click the checkbox Enable SQL HADR Service
- Click Ok on the warning dialog box

Step 2: Create an Availability Group

- Open SQL Server Management Studio on DENALI-SQL\INST1
- Select Management
- Right-click Availability Groups and select New Availability Group
- Click Next
- Give your new Availability Group a name and click Next
- Select which user databases you want to add to your Availability Group. Then click Next
- In the Specify Replicas screen, you can add the instances you want to be enable as HADR in the secondary role, after you have done this, click Next
- Next you have the overview screen, click Finish to start configuring the HADR setup
- click FINISH to finish the HADR setup, where you will get a progress screen

22 AlwaysOn Availability Groups

AlwaysOn Availability Groups provides a rich set of options that improve database availability and that enable improved resource use. The key components are as follows:

- Supports one primary replica and up to four secondary replicas.
- Supports alternative availability modes, as follows:
  - *Asynchronous-commit mode*. This availability mode is a disaster-recovery solution that works well when the availability replicas are distributed over considerable distances.
  - *Synchronous-commit mode*. This availability mode emphasizes high availability and data protection over performance, at the cost of increased transaction latency. A given availability group can support up to three synchronous-commit availability replicas, including the current primary replica.
- Enables you to configure a given availability replica to support either or both of the following capabilities when acting as a secondary replica:
  - Read-only access to the secondary databases.
  - Performing backup operations on secondary databases.

  By enabling these capabilities on your secondary replicas you can improve your IT efficiency and reduce cost through better resource utilization of secondary hardware. In addition, offloading read-intent applications and backup jobs to secondary replicas helps to improve performance on the primary replica.

- AlwaysOn Availability Groups provides an integrated set of tools to simplify deployment and management of availability groups, including:
  - The New Availability Group wizard, which simplifies availability-group creation.
  - The Availability Group Dashboard.
- Supports availability group listeners that provide fast application failover after an availability group fails over. Flexible Failover Policy for greater control over failover.
- Supports automatic page repair for protection against page corruption. Supports Encryption and compression, which provide a secure, high performing transport.

22.1 Overview of AlwaysOn Availability Groups

22.1.1 Availability Replicas and Roles

Each availability group defines a set of two or more failover partners known as availability replicas. *Availability replicas* are components of the availability group that are hosted by separate instances of SQL Server residing on different nodes of a WSFC failover cluster. Each of these server instances is either a failover cluster instance (FCI) or stand-alone instance on which you have enabled AlwaysOn Availability Groups. Each availability replica hosts a copy of the availability databases in the availability group.

Every availability replica is assigned an initial role—either the *primary role* or the *secondary role*, which is inherited by the availability databases of that replica. The role of a given replica determines whether it

hosts read-write databases or read-only databases. One replica, known as the *primary replica*, is assigned the primary role and hosts read-write databases, which are known as *primary databases*. At least one other replica, known as a *secondary replica*, is assigned the secondary role. A secondary replica hosts read-only databases, known as secondary databases. In this CTP, an availability group can contain only one secondary replica.

To add a database to an availability group, the database must be an online, read-write database that exists on the server instance that hosts the primary replica. When you add a database, it joins the availability group as a primary database, while remaining available to clients. No corresponding secondary database exists until backups of the new primary database are restored to the server instance that hosts the secondary replica (using RESTORE WITH NORECOVERY). The new secondary database is in the RESTORING state until it is joined to the availability group.

Joining places the secondary database into the ONLINE state and initiates data synchronization with the corresponding primary database. *Data synchronization* is the process by which changes to a primary database are reproduced on a secondary database. Data synchronization involves the primary replica sending transaction log records for the primary database to the secondary replica, which writes the transaction log records to disk (*hardens the log*) and applies the log records to the secondary database.

22.1.2 Availability Modes

The availability mode is a property of each availability replica. The availability mode determines whether the primary replica waits to commit transactions on a database until a given secondary replica has written the transaction log records to disk (hardened the log). AlwaysOn Availability Groups supports two availability modes—*asynchronous-commit mode* and *synchronous-commit mode*.

- **Asynchronous-commit mode**

  An availability replica that uses this availability mode is known as an *asynchronous-commit replica*. Under asynchronous-commit mode, the primary replica commits transactions without waiting for acknowledgement that an asynchronous-commit secondary replica has hardened the log. Asynchronous-commit mode minimizes transaction latency on the secondary databases but allows them to lag behind the primary databases, making some data loss possible.

- **Synchronous-commit mode**

  An availability replica that uses this availability mode is known as a *synchronous-commit replica*. Under synchronous-commit mode, before committing transactions, a synchronous-commit primary replica waits for a synchronous-commit secondary replica to acknowledge that it has finished hardening the log. Synchronous-commit mode ensures that once a given secondary database is synchronized with the primary database, committed transactions are fully protected. This protection comes at the cost of increased transaction latency.

22.1.3 Types of Failover

Within the context of a session between the primary replica and a secondary replica, the primary and secondary roles are potentially interchangeable in a process known as *failover*. During a failover the target secondary replica transitions to the primary role, becoming the new primary replica. The new primary replica brings its databases online as the primary databases, and client applications can connect to them. When the former primary replica is available, it transitions to the secondary role, becoming a secondary replica. The former primary databases become secondary databases and data synchronization resumes.

Three forms of failover exist—automatic, manual, and forced (with possible data loss). The form or forms of failover supported by a given secondary replica depends on its availability mode, and, for synchronous-commit mode, on the failover mode on the primary replica and target secondary replica, as follows.

Synchronous-commit mode supports two forms of failover—*manual failover* and *automatic failover*, if the target secondary replica is currently synchronized with the avt1. The support for these forms of failover depends on the setting of the *failover mode property* on the failover partners. If failover mode is set to "manual" on either the primary or secondary replica, only manual failover is supported for that secondary replica. If failover mode is set to "automatic" on both the primary and secondary replicas, both automatic and manual failover are supported on that secondary replica.

Manual failover (without data loss)

- A manual failover occurs after a database administrator issues a manual-failover command and causes a synchronized secondary replica to transition to the primary role (with guaranteed data protection) and the primary replica to transition to the secondary role. A manual failover requires that both the primary replica and the target secondary replica are running under synchronous-commit mode, and the secondary replica must already be synchronized.

Automatic failover (without data loss)

- An automatic failover occurs in response to a failure that causes a synchronized secondary replica to transition to the primary role (with guaranteed data protection). When the former primary replica becomes available, it transitions to the secondary role. Automatic failover requires that both the primary replica and the target secondary replica are running under synchronous-commit mode with the failover mode set to "Automatic". In addition, the secondary replica must already be synchronized, have WSFC quorum, and meet the conditions specified by the *flexible failover policy* of the availability group.
- Under asynchronous-commit mode, the only form of failover is forced failover (with possible data loss). Forced failover can only be initiated manually, and for this reason, is considered to be a kind of manual failover. However, forced failover is a disaster recovery option that is supported only when the secondary replica is not synchronized with the primary replica.

This is the only form of failover that is possible when the target secondary replica is not synchronized with the primary replica, even for synchronous-commit mode.

22.1.4 Allowing Read-Only Access to Secondary replicas

An availability replica can be configured so that, when performing the secondary role, it accepts client connections for read-only access to its local databases.

22.1.5 Client Connections

You can provide client connectivity to the primary replica of a given availability group by using one of the following methods:

- A network name

  You can create a network name for an availability group. To the current primary replica of the availability group, applications can specify this network name in connection strings. After the availability group fails over, the network name directs connections to the new primary replica. You must create a network name that is unique in the domain for each availability group. As you create a network name, a virtual IP (VIP) address is assigned to the network name.

  Only the TCP protocol is supported for using a network name and, optionally, a VIP to connect to an availability group. In the connection strings that your applications use to connect to the databases in the availability group, specify the availability group network name rather than the server name. Applications will connect directly to the current primary replica.

- Database-mirroring connection strings

  If an availability group possesses only two availability replicas and is not configured to allow read-access to the secondary replica, clients can connect to the primary replica by using database mirroring connection strings. This approach can be useful while migrating an existing application from database mirroring to an availability group, as long as you limit the availability group to two availability replicas. However, before you add additional availability replicas, you will need to create a VNN for the availability group and update your application to use the VNN.

  When using database mirroring connection strings, the client can use either SQL Server Native Client or .NET Framework Data Provider for SQL Server. The connection string provided by a client must minimally supply the name of one server instance, the *initial partner name*, to identify the server instance that initially hosts the availability replica to which you intend to connect. Optionally, the connection string can also supply the name of another server instance, the *failover partner name*, to identify the server instance that initially hosts the secondary replica as the failover partner name.

22.1.6 Automatic Page Repair

Each availability replica tries to automatically recover from corrupted pages on a local database by resolving certain types of errors that prevent reading a data page. If a secondary replica cannot read a page, the replica requests a fresh copy of the page from the primary replica. If the primary replica cannot read a page, the replica broadcasts a request for a fresh copy to all the secondary replicas and gets the page from the first to respond. If this request succeeds, the unreadable page is replaced by the copy, which usually resolves the error.

22.1.7 Interoperability and Coexistence with Other Database Engine Features

AlwaysOn Availability Groups can be used with the following features or components of SQL Server:

- Change data capture
- Change tracking
- Contained databases
- Database encryption
- Database snapshots
- FileStream and FileTable
- Full-text search
- Remote Blob Storage (RBS)
- Replication
- Service Broker
- SQL Server Agent

Refer: http://msdn.microsoft.com/en-us/library/ff877884%28v=SQL.110%29.aspx

23 Full-Text Search

23.1 Property Search

Beginning in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), you can configure a full-text index to support property-scoped searching on properties, such as Author and Title, which are emitted by IFilters. This form of searching is known as *property searching*. Whether property searching is possible on a given type of document depends on whether the corresponding filter (IFilter) extracts search properties during full-text indexing. Among IFilters that extract a number of document properties are the IFilters for Microsoft Office 2007 document file types, such as .docx, .xlsx, and .pptx.Ifilters are components that allow search services to index content of specific file type.

23.1.1 Known Restrictions on Property Searching

The following known restrictions on property searching exist:

- The XML IFilter does not emit properties.
- The IPropertyStore interface is not supported by SQL Server.

23.1.2 Search Property Lists

Property searching requires creating a *search property list* and specifying one or more properties that you want to make searchable. When you add a property to a search property list the property is registered for that particular list. To add a property to a search property list you need the following values:

- Property set GUID

  Each search property belongs to single property set that contains a group of related properties. Each property set is identified by a globally unique identifier (GUID).

- Property integer identifier

  Each search property possesses an identifier that is unique within the property set. Note that for a given property, the identifier could be either an integer or a string, however full-text search supports only integer identifiers.

- Property name

This is the name that users will specify in full-text queries to search on the property. A property name can contain internal spaces. The maximum length is 256 characters.

The property name can be any of the following:

The Windows canonical name of the property, such as System.Author or System.Contact.HomeAddress.

Overview of the CONTAINS Syntax for Property Searching

The basic CONTAINS syntax for a property-scoped full-text query is as follows:

SELECT *column_name* FROM *table_name*
WHERE
CONTAINS **(** PROPERTY (*column_name*, '*property_name*')**, '**<contains_search_condition>**')**;

23.1.3 Configuring a Full-Text Index for Property Searching

Adding a property to a search property list causes the Full-Text Engine to register the property for that particular property list. For a full-text index to support property searching on the properties that are registered for a search property list, you need to associate the search property list with the index and repopulate the index. Repopulating the full-text index creates property-specific index entries for search terms in each of the registered properties.

As long as the full-text index remains associated with this search property list, full-text query can use the PROPERTY option of the CONTAINS predicate to search on properties that are registered for that search property list.

23.1.4 Creating a Search Property List

23.1.4.1 *To create a search property list in Management Studio*

1. In Object Explorer, expand the server.
2. Expand Databases, and then expand the database in which you want to create the search property list.
3. Expand Storage, and then right-click Search Property Lists.
4. Select New Search Property List.
5. Specify the property list name.
6. Optionally, specify someone else as the property list owner.
7. Select one of the following options:
   - Create an empty search property list
   - Create from an existing search property list
8. Click OK.

23.1.4.2 *Viewing and Changing a Search Property List*

To view and change a search property list in Management Studio

1. In Object Explorer, expand the server.
2. Expand Databases, and then expand the database.
3. Expand Storage.
4. Expand Search Property Lists to display the search property lists.
5. Right-click the property list, and select Properties.
6. In the Search Property List Editor dialog box, use the Properties grid to add or remove search properties:
    a. To remove a document property, click the row header to the left of the property, and press DEL .
    b. To add a document property, click in the empty row at the bottom of the list, to the right of the *, and enter the values for the new property.
7. Click OK.

23.1.4.3 *Deleting a Search Property List*

You cannot drop a property list from a database, while the list is associated with any full-text index. For information about removing a search property list from a given full-text index, see How to: View and Change Properties of a Full-Text Index (SQL Server Management Studio).

To delete a search property list in Management Studio

1. In Object Explorer, expand the server.
2. Expand Databases, and then expand the database.
3. Expand Storage, and then expand the Search Property Lists node.
4. Right-click the property list that you want to delete, and click Delete.
5. Click OK

Refer: Search Document Properties with Search Property Lists.

23.2 Customizable NEAR

Beginning in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), you can customize a proximity search by using the new custom NEAR option of the CONTAINS predicate or CONTAINSTABLE function. Custom NEAR enables you to optionally specify the maximum number of non-search terms that separate the first and last search terms in a match. Custom NEAR also enables you to optionally specify that words and phrases are matched only if they occur in the order in which you specify them.

Searches for precise or fuzzy (less precise) matches to single words and phrases, words within a certain distance of one another, or weighted matches. CONTAINS is a predicate used in the WHERE clause of a Transact-SQL SELECT statement to perform SQL Server full-text search on full-text indexed columns containing character-based data types.

CONTAINS can search for:

- A word or phrase.
- The prefix of a word or phrase.
- A word near another word.
- A word inflectionally generated from another (for example, the word drive is the inflectional stem of drives, drove, driving, and driven).
- A word that is a synonym of another word using a thesaurus (for example, the word "metal" can have synonyms such as "aluminum" and "steel").

Refer: CONTAINS

CONTAINSTABL Returns a table of zero, one, or more rows for those columns containing precise or fuzzy (less precise) matches to single words and phrases, the proximity of words within a certain distance of one another, or weighted matches. CONTAINSTABLE is used in the FROM clause of a Transact-SQL SELECT statement and is referenced as if it were a regular table name. .It performs a SQL Server full-text search on full-text indexed columns containing character-based data types.

CONTAINSTABLE is useful for the same kinds of matches as the CONTAINS predicate and uses the same search conditions as CONTAINS.

Unlike CONTAINS, however, queries using CONTAINSTABLE return a relevance ranking value (RANK) and full-text key (KEY) for each row.

You can use a proximity term (NEAR) in a CONTAINS predicate or CONTAINSTABLE function to search for words or phrases near one another. You can also specify the maximum number of non-search terms that separate the first and last search terms. In addition, you can search for words or phrases in any order, or you can search for words and phrases in the order in which you specify them. Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) supports both the earlier generic proximity term, which is now deprecated, and the custom proximity term, which is new in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).

23.2.1 The Custom Proximity Term

The custom proximity term introduces the following new capabilities:

- You can specify the maximum number of non-search terms, or *maximum distance*, that separates the first and last search terms in order to constitute a match.
- If you specify the maximum number of terms, you can also specify that matches must contain the search terms in the specified order.

To qualify as a match, a string of text must:

- Start with one of the specified search terms and end with the one of the other specified search terms.
- Contain all of the specified search terms.
- The number of non-search terms, including stopwords, that occur between the first and last search terms must be less than or equal to the maximum distance, if specified.

The basic syntax is:

```
NEAR (
   {
     search_term [ ,...n ]
   |
     ( search_term [ ,...n ] ) [, <maximum_distance> [, <match_order> ] ]
   }
    )
```

23.2.2 How Maximum Distance Is Measured

A specific maximum distance, such as 10 or 25, determines how many non-search terms, including stopwords, can occur between the first and last search terms in a given string. For example, NEAR((dogs, cats, "hunting mice"), 3) would return the following row, in which the total number of non-search terms is three ("enjoy", "but", and "avoid"):

"Cats enjoy hunting mice, but avoid dogs."

The same proximity term would not return the following row, because the maximum distance is exceeded by the four non-search terms ("enjoy", "but", "usually", and "avoid"):

"Cats enjoy hunting mice, but usually avoid dogs."

23.2.3 Combining a Custom Proximity Term with Other Terms

You can combine a custom proximity term with some other terms. You can use AND (&), OR (|), or AND NOT (&!) to combine a custom proximity term with another custom proximity term, a simple term, or a prefix term. For example:

- CONTAINS('NEAR((*term1*,*term2*),5) AND *term3*')
- CONTAINS('NEAR((*term1*,*term2*),5) OR *term3*')
- CONTAINS('NEAR((*term1*,*term2*),5) AND NOT *term3*')
- CONTAINS('NEAR((*term1*,*term2*),5) AND NEAR((*term3*,*term4*),2)')
- CONTAINS('NEAR((*term1*,*term2*),5) OR NEAR((*term3*,*term4*),2, TRUE)')

Refer: Search for Words Close to Another Word with NEAR.

23.3 New Word Breakers and Stemmers

All the word breakers and stemmers used by Full-Text Search and Semantic Search, with the exception of the Korean language, are updated in this release. For consistency between the contents of indexes and the results of queries, we recommend that you repopulate existing full-text indexes after upgrading.

1. The third-party word breakers for English that were included with previous releases of SQL Server have been replaced with Microsoft components.

Refer: Change the Word Breaker Used for US English and UK English

2. The third-party word breakers for Danish, Polish, and Turkish that were included with previous releases of SQL Server have been replaced with Microsoft components. The new components are enabled by default.
3. There are new word breakers for Czech and Greek. Previous releases of SQL Server Full-Text Search did not include support for these two languages.
4. The behavior of the new word breakers has changed.

Refer: Behavior Changes to Full-Text Search.

24 Spatial data Type Enhancement

24.1 Circular Arc Segment Support for Spatial Types

Three new sub-data types for geometry and geography data types can be used to store circular arc segments

- CircularString
- CompoundCurve
- CurvePolygon

Methods for geography and geometry data types support the new circular arc segment data types. There are new methods for geometry and geography data types that work with circular arc segments

- OGC Methods on Geography Instances
- OGC Methods on Geometry Instances
- Extended Methods on Geography Instances
- Extended Methods on Geometry Instances
- SQL MM Methods on Geography Instances.

There are new static aggregate methods for geometry data type and geography data type

- Extended Static Geography Methods
- Extended Static Geometry Methods

A CircularString is a collection of zero or more continuous circular arc segments. A circular arc segment is a curved segment defined by three points in a two-dimensional plane; the first point cannot be the same as the third point. If all three points of a circular arc segment are collinear, the arc segment is treated as a line segment.

24.1.1 CircularString instances

Accepted instances

A CircularString instance is accepted if it is either empty or contains an odd number of points, n, where n > 1. T

Valid instances

A valid CircularString instance must be empty or have the following attributes:

- It must contain at least one circular arc segment (that is, have a minimum of three points).

- The last endpoint for each circular arc segment in the sequence, except for the last segment, must be the first endpoint for the next segment in the sequence.
- It must have an odd number of points.
- It cannot overlap itself over an interval.
- Although CircularString instances may contain line segments, these line segments must be defined by three collinear points.

Instances with collinear points

In the following cases a circular arc segment will be treated as a line segment:

- When all three points are collinear (for example, (1 3, 4 4, 7 5)).
- When the first and middle point are the same, but the third point is different (for example, (1 3, 1 3, 7 5)).
- When the middle and last point are the same, but the first point is different (for example, (1 3, 4 4, 4 4)).

Examples

A. Instantiating a Geometry Instance with an Empty CircularString

This example shows how to create an empty CircularString instance:

```
DECLARE @g geometry;

SET @g = geometry::Parse('CIRCULARSTRING EMPTY');
```

B. Instantiating a Geometry Instance Using a CircularString with One Circular Arc Segment

The following example shows how to create a CircularString instance with a single circular arc segment (half-circle):

```
DECLARE @g geometry;

SET @g = geometry:: STGeomFromText('CIRCULARSTRING(2 0, 1 1, 0 0)', 0);

SELECT @g.ToString ();
```

Z value or the same Z value.

24.1.2 CompoundCurve instances

Semicircle A uses two circular arc segments (ABC, CDA) to draw the perimeter. A CircularString instance could also store the perimeter of semicircle A.

Semicircle B uses one line segment (AB) and one circular arc segment (BCA) to draw the perimeter.

The following example shows how a CompoundCurve instance can store the perimeter of Semicircle B in a clear concise manner.

```
SET @g = geometry::Parse('CompoundCurve((0 1, 2 1), CIRCULARSTRING(2 1, 1 2, 0 1))');
```

The following example shows how a CircularString instance would the semicircle.

```
SET @g= geometry::Parse('CIRCULARSTRING(0 1, 2 1, 2 1, 1 2, 0 1)');
```

The CompoundCurve format distinguishes the line segment from the circular arc segment of the semicircle.

Accepted instances

CompoundCurve instance is accepted if it is an empty instance or meets the following criteria.

1. All the instances contained by CompoundCurve instance are accepted circular arc segment instances. For more information on accepted circular arc segment instances, see LineString and CircularString.
2. All of the circular arc segments in the CompoundCurve instance are connected. The first point for each succeeding circular arc segment is the same as the last point on the preceeding circular arc segment.
3. None of the contained instances are empty instances.

Valid instances

A CompoundCurve instance is valid if it meets the following criteria.

1. The CompoundCurve instance is accepted.
2. All circular arc segment instances contained by the CompoundCurve instance are valid instances.

Examples

A. Instantiating a geometry instance with an empty CompooundCurve

The following example shows how to create an empty CompoundCurve instance:

```
DECLARE @g geometry;

SET @g = geometry::Parse('COMPOUNDCURVE EMPTY');
```

24.1.3 CurvePolygon

A CurvePolygon is a topologically closed surface defined by an exterior bounding ring and zero or more interior rings

The following criteria define attributes of a CurvePolygon instance:

- The boundary of the CurvePolygon instance is defined by the exterior ring and all interior rings.
- The interior of the CurvePolygon instance is the space between the exterior ring and all of the interior rings.

A CurvePolygon instance differs from a Polygon instance in that a CurvePolygon instance may contain the following circular arc segments: CircularString and CompoundCurve.

CompoundCurve instances

Accepted instances

For a CurvePolygon instance to be accepted, it needs to be either empty or contain only circular arc rings that are accepted. An accepted circular arc ring meets the following requirements.

1. Is an accepted LineString, CircularString, or CompoundCurve instance.
2. Has at least four points.
3. The start and end point have the same X and Y coordinates.

Valid instances

For a CurvePolygon instance to be valid both exterior and interior rings must meet the following criteria:

1. They may only touch at single tangent points.
2. They cannot cross each other.
3. Each ring must contain at least four points.
4. Each ring must be an acceptable curve type.

CurvePolygon instances also need to meet specific criteria depending on whether they are geometry or geography data types.

Geometry data type

A valid geometryCurvePolygon instance must have the following attributes:

1. All the interior rings must be contained within the exterior ring.
2. May have multiple interior rings, but an interior ring cannot contain another interior ring.

3. No ring can cross itself or another ring.
4. Rings can only touch at single tangent points (number of points where rings touch must be finite).
5. The interior of the polygon must be connected.

Geography data type

A valid geographyCurvePolygon instance must have the following attributes:

1. The interior of the polygon is connected using the left-hand rule.
2. No ring can cross itself or another ring.
3. Rings can only touch at single tangent points (number of points where rings touch must be finite).
4. The interior of the polygon must be connected.

Examples

A. Instantiating a Geometry Instance with an Empty CurvePolygon

This example shows how to create an empty CurvePolygon instance:

```
DECLARE @g geometry;
SET @g = geometry::Parse('CURVEPOLYGON EMPTY');
```

24.2 New methods for geometry and geography data types

24.2.1 OGC Methods on Geography Instances

- STArea
- STAsBinary
- STAsText
- STBuffer
- STCurveN (geography Data Type)
- STCurveToLine (geography Data Type)
- STDifference
- STDimension
- STDisjoint
- STDistance
- STEndpoint
- STEquals
- STGeometryN
- STGeometryType
- STIntersection
- STIntersects

- STIsClosed
- STIsEmpty
- STIsValida
- STLength
- STNumCurves (geography Data Type)
- STNumGeometries
- STNumPoints
- STPointN
- STSrid
- STStartPoint
- STSymDifference
- STUnion

Refer: http://msdn.microsoft.com/en-us/library/bb933917(v=sql.110).aspx

24.2.2 Extended Methods on Geography Instances

- AsGml
- AsTextZM
- BufferWithCurves (geography Data Type)
- BufferWithTolerance
- CurveToLineWithTolerance (geography Data Type)
- EnvelopeAngle
- EnvelopeCenter
- Filter
- InstanceOf
- IsNull
- Lat
- Long
- M
- MakeValida
- MinDbCompatibilityLevel (geography Data Type)
- NumRing
- Reduce
- ReorientObject
- RingN
- ShortestLineTo (geography Data Type)
- ToString
- Z

Refer: http://msdn.microsoft.com/en-us/library/bb933968(v=sql.110).aspx

24.2.3 OGC Methods on Geometry Instances

- STArea

- STAsBinary
- STAsText
- STBoundary
- STBuffer
- STCentroid
- STContains
- STConvexHull
- STCrosses
- STCurveN (geometry Data Type)
- STCurveToLine (geometry Data Type)
- STDifference
- STDimension
- STDisjoint
- STDistance
- STEndpoint
- STEnvelope
- STEquals
- STExteriorRing http://msdn.microsoft.com/en-us/library/bb933960(v=sql.110).aspx
- STGeometryN
- STGeometryType
- STInteriorRingN
- STIntersection
- STIntersects
- STIsClosed
- STIsEmpty
- STIsRing
- STIsSimple
- STIsValid
- STLength
- STNumCurves (geometry Data Type)
- STNumGeometries
- STNumInteriorRing
- STNumPoints
- STOverlaps
- STPointN
- STPointOnSurface
- STRelate
- STSrid
- STStartPoint
- STSymDifference
- STTouches
- STUnion
- STWithin
- STX
- STY

Refer: http://msdn.microsoft.com/en-us/library/bb933960(v=sql.110).aspx

24.2.4 Extended Methods on Geometry Instances

[SQL Server supports a number of extended methods on Open Geospatial Consortium (OGC) geometry instances.

- AsGml (geometry Data Type)
- AsTextZM (geometry Data Type)
- BufferWithCurves (geometry Data Type)
- BufferWithTolerance (geometry Data Type)
- CurveToLineWithTolerance (geometry Data Type)
- InstanceOf (geometry Data Type)
- Filter (geometry Data Type)
- IsNull (geometry Data Type)
- M (geometry Data Type)
- MakeValid (geometry Data Type)
- MinDbCompatibilityLevel (geometry Data Type)
- Reduce (geometry Data Type)
- ShortestLineTo (geography Data Type)
- ToString (geometry Data Type)
- Z (geometry Data Type)

Refer: http://msdn.microsoft.com/en-us/library/bb933968(v=sql.110).aspx

24.2.5 SQL MM Methods on Geography Instances

SQL Server supports the SQL/MM Spatial standard (ISO/IEC 13249-3:201x) methods on geography instances.

- STContains (geography Data Type)
- STConvexHull (geography Data Type)
- STOverlaps (geography Data Type)
- STWithin (geography Data Type)

Refer: http://msdn.microsoft.com/en-us/library/ff929170(v=sql.110).aspx

24.3 New static aggregate methods for geometry data type and geography data type

24.3.1 Extended Static Geography Methods

SQL Server supports several extensions to the static geography methods of the Open Geospatial Consortium (OGC).

- CollectionAggregate (geography Data Type)
- ConvexHullAggregate (geography Data Type)
- EnvelopeAggregate (geography Data Type)
- GeomFromGML
- Null
- Parse
- Point
- UnionAggregate (geography Data Type)

Refer: http://msdn.microsoft.com/en-us/library/bb933921(v=sql.110).aspx

24.3.2 Extended Static Geometry Methods

SQL Server supports several extensions to the static geometry methods of the Open Geospatial Consortium (OGC).

- CollectionAggregate (geometry Data Type)
- ConvexHullAggregate (geometry Data Type)
- EnvelopeAggregate (geometry Data Type)
- GeomFromGML
- Null
- Parse
- Point
- UnionAggregate (geometry Data Type)

Refer: http://msdn.microsoft.com/en-us/library/bb933805(v=sql.110).aspx

25 Service Broker

SQL Server Service Broker includes the following:

- Support for conversation priorities
- A new command prompt utility to diagnose Service Broker configurations and conversations
- A new performance object and counters
- Support for Service Broker in SQL Server Management Studio
- New tutorials.

25.1 New Conversation Priorities

Conversation priorities let administrators and developers specify that messages for important Service Broker conversations are sent and received before messages from less important conversations. This ensures that low priority work does not block higher priority work. Service Broker systems can be configured to offer varying levels of service

25.1.1 Conversation Priorities

Conversation priorities are a set of user-defined rules, each of which specifies a priority level and the criteria for determining which Service Broker conversations to assign the priority level. Messages from conversations that have high priority levels are typically sent or received before messages from conversations that have low priority levels.

- For the target conversation endpoint, the target service is the local service and the initiator service is the remote service.

25.1.2 How Service Broker Assigns Priority Levels

Service Broker assigns conversation priority levels when conversation endpoints are created. The conversation endpoint retains the priority level until the conversation ends. New priorities or changes to existing priorities are not applied to existing conversations.

Service Broker assigns the conversation endpoint the priority level from the conversation priority whose contract and services criteria best match the endpoint properties. The following table shows the match precedence:

Endpoint Contract	Endpoint Local Service	Endpoint Remote Service
Priority Contract	Priority Local Service	Priority Remote Service
Priority Contract	Priority Local Service	ANY

Priority Contract	ANY	Priority Remote Service
Priority Contract	ANY	ANY
ANY	Priority Local Service	Priority Remote Service
ANY	Priority Local Service	ANY
ANY	ANY	Priority Remote Service
ANY	ANY	ANY

Service Broker first looks for a priority whose specified contract, local service, and remote service matches those used by the conversation endpoint. If one is not found, Service Broker then looks for a priority with a contract and local service that matches those used by the endpoint, and where the remote service was specified as ANY. This continues for all the variations listed in the precedence table. If no match is found, the endpoint is assigned the default priority of 5.

The Service Broker communication protocols do not transmit priority levels between conversation endpoints. Service Broker independently assigns a priority level to each endpoint. To have Service Broker assign priority levels to both the initiator and target conversation endpoints, you must ensure that both endpoints are covered by conversation priorities. If the initiator and target conversation endpoints are in separate databases, you must create conversation priorities in each database. If the initiator and target endpoints are in the same database:

- You can cover both conversation endpoints by using one conversation priority that specifies the contract name that is used by the conversation and ANY for both the local and remote service names.
- You can cover each conversation endpoint separately using two conversation priorities:
  - One conversation for the initiator endpoint that specifies the initiator service name for LOCAL_SERVICE_NAME and the target service name for REMOTE_SERVICE_NAME.
  - One conversation for the target endpoint that specifies the target service name for LOCAL_SERVICE_NAME and the initiator service name for REMOTE_SERVICE_NAME.

The same priority level is usually specified for both of the conversation endpoints for a conversation. While you can specify different priority levels for each endpoint, doing this does not mean messages are sent faster in one direction than the other. Messages are sent from one conversation endpoint and received at the other endpoint. Therefore, each message transmission is affected by the priority levels assigned to both endpoints. For example, you could configure a conversation so that the initiator conversation endpoint has priority level 10 and the target endpoint has priority level 1. In this case:

- Messages transmitted from the initiator service by using priority level 10 are received from the target queue using priority level 1.
- Messages transmitted from the target service by using priority level 1 are received from the initiator queue using priority level 10.

A conversation group is assigned the same priority level as the highest priority level assigned to any conversation where the following are true:

- The conversation is a member of the group.
- The conversation currently has messages in the service queue.

All conversation endpoints in a database are assigned default priorities of 5 if no conversation priorities have been created in the database.

Conversation priorities do not affect message forwarding, which always operates at the default priority level of 5.

25.2 New Diagnostic Utility

The ssbdiagnose utility analyzes the configuration between two Service Broker services, or for a single service. The utility also analyzes running conversations for errors. If a running conversation encounters errors, ssbdiagnose analyzes the Service Broker configuration that is used by the conversation. Errors are reported either in the command prompt window as human-readable text, or as formatted XML that can be redirected to a file or another program.

25.2.1 ssbdiagnose Utility

The ssbdiagnose utility reports issues in Service Broker conversations or the configuration of Service Broker services. Configuration checks can be made for either two services or a single service. Issues are reported either in the command prompt window as human-readable text, or as formatted XML that can be redirected to a file or another programanother program.

25.3 New Service Broker Elements in Object Explorer

Conversation priorities have been added to the SQL Server Management Studio Object Explorer hierarchy. Existing Service Broker objects have additional right-click menu items, including Properties menu items

25.4 New System Monitor Object and Counters

The Broker TO Statistics performance object reports how often Service Broker dialogs request transmission objects and how often inactive transmission objects are written to work tables in tempdb.

The following five new counters have been added to the Broker Statistics performance object:

- Activation Errors Total
- Corrupted Messages Total
- Dequeued TransmissionQ Msgs/sec
- Dropped Messages Total
- Enqueued TransmissionQ Msgs/sec

25.5 New Service Broker Tutorials

Three Service Broker tutorials have been added to illustrate the steps that are required to set up simple request-reply conversations with three scopes:

- The initiator and target services are in the same database.
- The initiator and target services are in separate databases in the same instance of the Database Engine.
- The initiator and target services are in separate instances.

There is also an Activation tutorial that illustrates the steps that are required to configure an activation stored procedure to receive messages from a queue.

25.5.1 Service Broker Tutorials

SQL Server Service Broker is used to create conversations for exchanging messages. Messages are exchanged between two ends: the target and initiator. Messages are used to transmit data, and to trigger processing when a message is received. The target and the initiator can be in the same database, different databases on the same instance of the Database Engine, or on separate instances. All tutorials are for users who are new to Service Broker. To start a tutorial, click one of the following links:

Completing a Conversation in a Single Database .

Shows you how to create the objects that are required to support a simple conversation. Then, you start a conversation to send and receive messages.

Completing a Conversation Between Databases

Builds on the first Service Broker tutorial. It shows you how to extend a conversation between two databases in the same instance of the Database Engine.

Completing a Conversation Between Instances

Builds on the first Service Broker tutorial. It shows you how to extend a conversation between databases in separate instances of the Database Engine.

Implementing Internal Activation

Builds on the first Service Broker tutorial. It shows you how to create a stored procedure to receive messages from a queue, and use internal activation to run the stored procedure whenever there are messages in the queue.

Refer: http://msdn.microsoft.com/en-us/library/bb522897(v=sql.110).aspx

26 Business Intelligence

Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3) Project Crescent is an interactive data exploration, visualization, and presentation experience. It provides drag-and-drop ad hoc reporting for business users such as data analysts, business decision makers, and information workers. Project Crescent reports are in a new file format, RDLX.

You can select text and then drag and drop it in another location. You can drag and drop text:

- From Object Explorer into the editor to create a query.
- From one location to a different location in the current editor.
- Between editors.
- To the Microsoft Windows Recycle Bin.

To drag and drop text

1. Select the text you want to move, either with the mouse or with the keyboard.
2. Left-click the highlighted text and continue holding down the mouse button.
3. Move the mouse cursor to the destination where you want to place the text.
4. Release the mouse button to drop the text.

Dragging text moves it; that is, the text is erased from the old location and moved to the new location. Pressing the CTRL key while dragging text copies it to the new location. Dragging an object from Object Explorer is an exception and does not delete the object name from Object Explorer.

Project Crescent expands on the self-service BI capabilities delivered with PowerPivot for Excel and PowerPivot for SharePoint by enabling customers to visualize and interact with modeled data in a meaningful way, using interactive visualizations, animations, and smart querying. It is a browser-based Silverlight application launched from within SharePoint Server 2010 that enables users to present and share insights with others in their organization through interactive presentations.

26.1 Based on Tabular Models

With Project Crescent, customers start from an SQL Server Code-Named "Denali" Analysis Services (SSAS), Community Technology Preview 3 (CTP 3) tabular model to build their reports. Tabular models

use metadata to present an underlying data source to end users, with predefined relationships and behaviors, in terms they understand.

26.2 Coexists with Report Builder

Project Crescent does not replace Report Builder, the report authoring tool for richly designed operational reports. Project Crescent addresses the need for Web-based, ad hoc reporting. It co-exists with the latest version of Report Builder, which also ships in SQL Server Code-Named “Denali.”

26.3 Author Tabular Models in Business Intelligence Development Studio

The Tabular Model Designer, first introduced in CTP2, is now integrated with Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) Business Intelligence Development Studio.

Also included with this release is the Tabular Modeling Adventure Works Tutorial. This tutorial guides BI software professionals through creating a new tabular model project in Business Intelligence Development Studio, importing data from the AdventureWorksDWDenali sample database, adding relationships, calculations, perspectives, roles, and hierarchies, and then deploying the model.

26.3.1 Tabular Modeling (SSAS)

Tabular models are in-memory databases in Analysis Services. Using state-of-the-art compression algorithms and multi-threaded query processor, the VertiPaq™ engine delivers fast access to tabular model objects and data by reporting client applications such as Microsoft Excel and Microsoft Project Crescent.

Tabular models support data access through two modes: Cached mode and DirectQuery mode. In cached mode, you can integrate data from multiple sources including relational databases, data feeds, and flat text files. In DirectQuery mode, you can bypass the in-memory model, allowing client applications to query data directly at the (SQL Server relational) source.

Tabular models are authored in Business Intelligence Development Studio using new tabular model project templates. You can import data from multiple sources, and then enrich the model by adding relationships, calculated columns, measures, KPIs, and hierarchies. Models can then be deployed to an instance of Analysis Services where client reporting applications can connect to them. Deployed models can be managed in SQL Server Management Studio just like multidimensional models. They can also be partitioned for optimized processing and secured to the row-level by using role based security.

26.3.2 Tabular Model Solutions (SSAS)

Topics in this section provide information about how to author tabular models by using the tabular model designer in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) Business Intelligence Development Studio. Please refer the links below to learn about the tabular model.

- Tabular Model Designer (SSAS)
- Workspace Database (SSAS)
- Tabular Model Projects (SSAS)
- Properties (SSAS)
- Data Sources (SSAS)
- Tables and Columns (SSAS)
- Relationships (SSAS - Tabular Models)
- Calculations (SSAS - Tabular Models)
- Measures (SSAS -Tabular Models)
- KPIs (SSAS -Tabular Models)
- Hierarchies (SSAS -Tabular Models)
- Partitions (SSAS -Tabular Models)
- Perspectives (SSAS -Tabular Models)
- Roles (SSAS -Tabular Models)
- DirectQuery Mode (SSAS)
- Analyze in Excel (SSAS -Tabular Models)
- Tabular Model Solution Deployment (SSAS)

26.4 Tabular Modeling (Adventure Works Tutorial)

This tutorial provides lessons on how to create a Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) Analysis Services tabular model running in Cached mode by using Business Intelligence Development Studio.

- How to create a new tabular model project in BI Development Studio.
- How to import data from a SQL Server relational database into a tabular model project.
- How to create and manage relationships between tables in the model.
- How to create and manage calculations, measures, and Key Performance Indicators that help users analyze model data.
- How to create and manage perspectives and hierarchies that help users more easily browse model data by providing business and application specific viewpoints.
- How to create partitions that divide table data into smaller logical parts that can be processed independent from other partitions.
- How to secure model objects and data by creating roles with user members.
- How to deploy a tabular model in Cached mode to a sandbox or production instance of Analysis Services.

Prerequisites

In order to complete this tutorial, you must have the following prerequisites installed:

- Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) Analysis Services (running in Tabular mode).
- Business Intelligence Development Studio - installed as part of Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).
- AdventureWorksDWDenali sample database. This sample database includes the data necessary to complete this tutorial. To download the sample database, see http://go.microsoft.com/fwlink/?LinkID=220093.
- Microsoft Excel 2003 or later (for use with the Analyze in Excel feature in lesson 11)

Please refer the below link for adventure works tutorial on how to create a tabular model running in Cached mode by using Business Intelligence Development Studio.

Refer: **Tabular Modeling (Adventure Works Tutorial)**

27 Online Index Rebuild

Indexes containing varchar (max), nvarchar (max), and varbinary (max) columns can now be rebuilt as an online operation.

27.1 Online Index Operations

Indexes can be created, rebuild, or dropped online. The ONLINE option allows concurrent user access to the underlying table or clustered index data and any associated nonclustered indexes during these index operations. For example, while a clustered index is being rebuilt by one user, that user and others can continue to update and query the underlying data. While performing DDL operations offline, such as building or rebuilding a clustered index; these operations hold exclusive locks on the underlying data and associated indexes. This prevents modifications and queries to the underlying data until the index operation is complete.

It is recommended to perform online index operations for business environments that operate 24 hours a day, seven days a week, in which the need for concurrent user activity during index operations is vital.

The ONLINE option is available in the following Transact-SQL statements.

- CREATE INDEX

- ALTER INDEX
- DROP INDEX
- ALTER TABLE

Example

In the following example, all indexes on the Product table in the AdventureWorks sample database are rebuilt online.

```
USE AdventureWorks;
GO
ALTER INDEX ALL ON Production.Product
REBUILD WITH (ONLINE = ON);
```

27.2 How Online Index Operations Work

This topic defines the structures that exist during an online index operation and shows the activities associated with these structures.

27.2.1 Online Index Structures

To allow for concurrent user activity during an index data definition language (DDL) operation, the following structures are used during the online index operation: source and preexisting indexes, target, and for clustered indexes, a temporary mapping index.

- Source and preexisting indexes

  The source is the original table or clustered index data. Preexisting indexes are any nonclustered indexes that are associated with the source structure. For example, if the online index operation is rebuilding a clustered index that has four associated nonclustered indexes, the source is the existing clustered index and the preexisting indexes are the nonclustered indexes.

  The preexisting indexes are available to concurrent users for select, insert, update, and delete operations. This includes bulk inserts (supported but not recommended) and implicit updates by triggers and referential integrity constraints. All preexisting indexes are available for queries and searches. This means they may be selected by the query optimizer and, if necessary, specified in index hints.

- Target

The target or targets is the new index (or heap) or a set of new indexes that is being created or rebuilt. User insert, update, and delete operations to the source are applied by the SQL Server Database Engine to the target during the index operation. For example, if the online index operation is rebuilding a clustered index, the target is the rebuilt clustered index; the Database Engine does not rebuild nonclustered indexes when a clustered index is rebuilt.

The target index is not searched while processing SELECT statements until the index operation is committed. Internally, the index is marked as write-only.

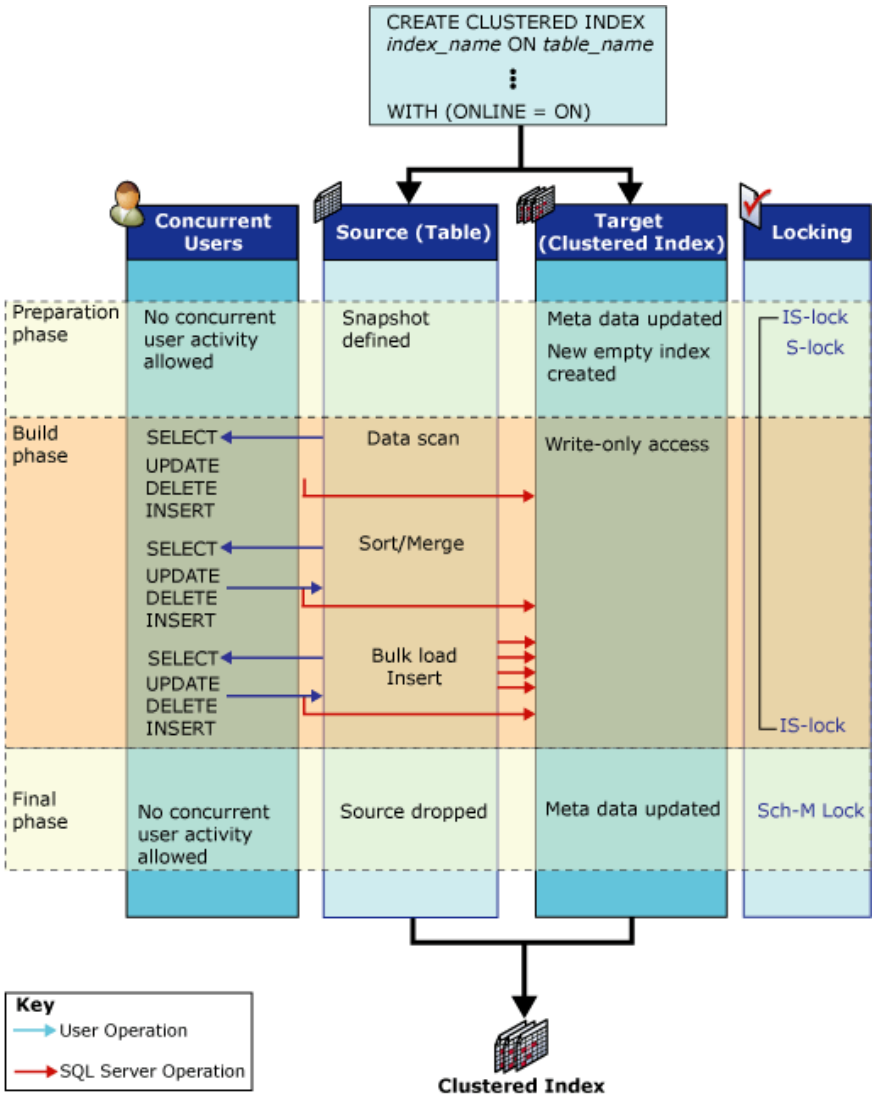
- Temporary mapping index

  Online index operations that create, drop, or rebuild a clustered index also require a temporary mapping index. This temporary index is used by concurrent transactions to determine which records to delete in the new indexes that are being built when rows in the underlying table are updated or deleted. This nonclustered index is created in the same step as the new clustered index (or heap) and does not require a separate sort operation. Concurrent transactions also maintain the temporary mapping index in all their insert, update, and delete operations.

27.2.2 Online Index Activities

During a simple online index operation, such as creating a clustered index on a nonindexed table (heap), the source and target go through three phases: preparation, build, and final.

The following illustration shows the process for creating an initial clustered index online. The source object, the heap, has no other indexes. The source and target structure activities are shown for each phase; and concurrent user select, insert, update, and delete operations are also shown. The preparation, build, and final phases are indicated together with the lock modes used in each phase.



27.2.3 Source Structure Activities

The following table lists the activities involving the source structures during each phase of the index operation and the corresponding locking strategy.

Phase	Source activity	Source locks

Preparation Very short phase	System metadata preparation to create the new empty index structure. A snapshot of the table is defined. That is, row versioning is used to provide transaction-level read consistency. Concurrent user write operations on the source are blocked for a very short period. No concurrent DDL operations are allowed except creating multiple nonclustered indexes.	S (Shared) on the table* IS (Intent Shared) INDEX_BUILD_INTERNAL_RESOURCE**
Build Main phase	The data is scanned, sorted, merged, and inserted into the target in bulk load operations. Concurrent user select, insert, update, and delete operations are applied to both the preexisting indexes and any new indexes being built.	IS (Intent Shared) INDEX_BUILD_INTERNAL_RESOURCE**
Final Very short phase	All uncommitted update transactions must complete before this phase starts. Depending on the acquired lock, all new user read or write transactions are blocked for a very short period until this phase is completed. System metadata is updated to replace the source with the target. The source is dropped if it is required. For example, after rebuilding or dropping a	INDEX_BUILD_INTERNAL_RESOURCE** S (Shared) on the table if creating a nonclustered index.* SCH-M (Schema Modification) if any source structure (index or table) is dropped.*

	clustered index.	

* The index operation will wait for any uncommitted update transactions to complete before acquiring the S-lock or SCH-M lock on the table.

** The resource lock INDEX_BUILD_INTERNAL_RESOURCE prevents the execution of concurrent DDL operations on the source and preexisting structures while the index operation is in progress. For example, this lock prevents concurrent rebuild of two indexes on the same table. Although this resource lock is associated with the Sch-M lock, it does not prevent data manipulation statements.

The previous table shows a single Shared (S) lock acquired during the build phase of an online index operation that involves a single index. When clustered and nonclustered indexes are built, or rebuilt, in a single online index operation (for example, during the initial clustered index creation on a table that contains one or more nonclustered indexes) two short-term Shared (S) locks are acquired during the build phase followed by long-term IS locks. One S lock is acquired first for the clustered index creation and when creating the clustered index is completed, a second short-term S lock is acquired for creating the nonclustered indexes. After the nonclustered indexes are created, the S lock is downgraded to an IS lock until the final phase of the online index operation.

27.2.4 Target Structure Activities

The following table lists the activities that involve the target structure during each phase of the index operation and the corresponding locking strategy.

Phase	Target activity	Target locks
Preparation	New index is created and set to write-only.	IS (Intent Shared)
Build	Data is inserted from source. User modifications (inserts, updates, deletes) applied to the source are applied. This activity is transparent to the user.	IS (Intent Shared)
Final	Index metadata is updated.	S (Shared)

	Index is set to read/write status.	or SCH-M (Schema Modification)

The target is not accessed by SELECT statements issued by the user until the index operation is completed.

After the preparation and final phase is completed, the query and update plans that are stored in the procedure cache are invalidated. Subsequent queries will use the new index.

The lifetime of a cursor declared on a table that is involved in an online index operation is limited by the online index phases. Update cursors are invalidated at each phase. Read-only cursors are invalidated only after the final phase.

27.3 Guidelines for Online Index Operations

- Clustered indexes must be created, rebuilt, or dropped offline when the underlying table contains the following large object (LOB) data types: image, ntext, text, and xml.
- Nonunique nonclustered indexes can be created online when the table contains LOB data types but none of these columns are used in the index definition as either key or nonkey (included) columns. Nonclustered indexes defined with LOB data type columns must be created or rebuilt offline.
- Indexes on local temp tables cannot be created, rebuilt, or dropped online. This restriction does not apply to indexes on global temp tables.
- Online index operations are available only in SQL Server Enterprise, Developer, and Evaluation editions.

The following table shows the index operations that can be performed online and the indexes that are excluded from these online operations. Additional restrictions are also included.

Online index operation	Excluded indexes	Other restrictions
ALTER INDEX REBUILD	Disabled clustered index or disabled indexed view	Specifying the keyword ALL may cause the operation to fail when the table contains an excluded index.

	XML index Index on a local temp table	Additional restrictions on rebuilding disabled indexes apply.
CREATE INDEX	XML index Initial unique clustered index on a view Index on a local temp table	
CREATE INDEX WITH DROP_EXISTING	Disabled clustered index or disabled indexed view Index on a local temp table XML index	
DROP INDEX	Disabled index XML index Nonclustered index Index on a local temp table	Multiple indexes cannot be specified within a single statement.
ALTER TABLE ADD CONSTRAINT (PRIMARY KEY or UNIQUE)	Index on a local temp table Clustered index	Only one subclause is allowed at a time. For example, you cannot add and drop PRIMARY KEY or UNIQUE constraints in the same ALTER TABLE statement.
ALTER TABLE DROP CONSTRAINT (PRIMARY KEY or UNIQUE)	Clustered index	

The underlying table cannot be modified, truncated, or dropped while an online index operation is in process.

The online option setting (ON or OFF) specified when you create or drop a clustered index is applied to any nonclustered indexes that must be rebuilt. For example, if the clustered index is built online by using CREATE INDEX WITH DROP_EXISTING, ONLINE=ON, all associated nonclustered indexes are re-created online also.

When you create or rebuild a UNIQUE index online, the index builder and a concurrent user transaction may try to insert the same key, therefore violating uniqueness. If a row entered by a user is inserted into the new index (target) before the original row from the source table is moved to the new index, the online index operation will fail.

Although not common, the online index operation can cause a deadlock when it interacts with database updates because of user or application activities. In these rare cases, the SQL Server Database Engine will select the user or application activity as a deadlock victim.

You can perform concurrent online index DDL operations on the same table or view only when you are creating multiple new nonclustered indexes, or reorganizing nonclustered indexes. All other online index operations performed at the same time fail. For example, you cannot create a new index online while rebuilding an existing index online on the same table.

27.3.1 Disk Space Considerations

Generally, disk space requirements are the same for online and offline index operations. An exception is additional disk space required by the temporary mapping index. This temporary index is used in online index operations that create, rebuild, or drop a clustered index. Dropping a clustered index online requires as much space as creating a clustered index online.

27.3.2 Performance Considerations

Although online index operations permit concurrent user update activity, the index operations will take longer if the update activity is very heavy. Typically, online index operations will be slower than equivalent offline index operations regardless of the concurrent update activity level.

Because both the source and target structures are maintained during the online index operation, the resource usage for insert, update, and delete transactions is increased, potentially up to double. This could cause a decrease in performance and greater resource usage, especially CPU time, during the index operation. Online index operations are fully logged.

Although we recommend online operations, you should evaluate your environment and specific requirements. It may be optimal to run index operations offline. In doing this, users have restricted access to the data during the operation, but the operation finishes faster and uses fewer resources.

On multiprocessor computers that are running SQL Server Enterprise, index statements may use more processors to perform the scan and sort operations associated with the index statement just like other queries do. You can use the MAXDOP index option to control the number of processors dedicated to the online index operation. In this way, you can balance the resources that are used by index operation with those of the concurrent users.

Because an S-lock or Sch-M lock is held in the final phase of the index operation, be careful when you run an online index operation inside an explicit user transaction, such as BEGIN TRANSACTION...COMMIT block. Doing this causes the lock to be held until the end of the transaction, therefore impeding user concurrency.

27.3.3 Transaction Log Considerations

Large-scale index operations, performed offline or online, can generate large data loads that can cause the transaction log to quickly fill. To make sure that the index operation can be rolled back, the transaction log cannot be truncated until the index operation has been completed; however, the log can be backed up during the index operation. Therefore, the transaction log must have sufficient space to store both the index operation transactions and any concurrent user transactions for the duration of the index operation.

28 IT Administration

28.1 Provisioning During Setup

To enhance role separation, BUILTIN\administrators and Local System (NT AUTHORITY\SYSTEM) are not automatically provisioned in the sysadmin fixed server role. Local administrators can still access the Database Engine when in single user mode.

SQL Server now supports Managed Service Accounts and Virtual Accounts when installed on Windows 7 or Windows Server 2008 R2. For more information, see Configure Windows Service Accounts and Permissions.

The protection of operating services under a per-service SID is now extended to all operating systems. For more information, see Configure Windows Service Accounts and Permissions.

28.2 New Permissions

New GRANT, REVOKE, and DENY permissions to a SEARCH PROPERTY LIST are available.

New GRANT, REVOKE, and DENY permissions to CREATE SERVER ROLE and ALTER ANY SERVER ROLE.

28.3 New Role Management

User-defined server roles are now available to manage user-defined server roles and to add and remove members from all server roles.

28.3.1 Create Server Role (Transact-SQL)

Creates a new user-defined server role.User-defined server roles are new in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).

Syntax

CREATE SERVER ROLE role_name [AUTHORIZATION server_principal]

Arguments

- Role_name Is the name of the server role to be created.
- AUTHORIZATION server_principal Is the login that will own the new server role. If no login is specified, the server role will be owned by the login that executes CREATE SERVER ROLE.

Server roles are server-level securables. After you create a server role, configure the server-level permissions of the role by using GRANT, DENY, and REVOKE. To add logins to or remove logins from a server role, use ALTER SERVER ROLE (Transact-SQL). To drop a server role, use DROP SERVER ROLE (Transact-SQL).

You can view the server roles by querying the sys.server_role_members and sys.server_principals catalog views.

Server roles cannot be granted permission on database-level securables. To create database roles, see CREATE ROLE (Transact-SQL).

Permissions

Requires CREATE SERVER ROLE permission or membership in the sysadmin fixed server role.

Also requires IMPERSONATE on the *server_principal* for logins, ALTER permission for server roles used as the *server_principal*, or membership in a Windows group that is used as the server_principal.

This will fire the Audit Server Principal Management event with the object type set to server role and event type to add.

When you use the AUTHORIZATION option to assign server role ownership, the following permissions are also required:

- To assign ownership of a server role to another login, requires IMPERSONATE permission on that login.
- To assign ownership of a server role to another server role, requires membership in the recipient server role or ALTER permission on that server role.

Examples

A. Creating a server role that is owned by a login

The following example creates the server role buyers that is owned by login BenMiller.

```
USE master;
CREATE SERVER ROLE buyers AUTHORIZATION BenMiller;
GO
```

B. Creating a server role that is owned by a fixed server role

The following example creates the server role auditors that is owned the securityadmin fixed server role.

```
USE master;
CREATE SERVER ROLE auditors AUTHORIZATION securityadmin;
GO
```

28.3.2 ALTER SERVER ROLE (Transact-SQL)

Changes the membership of a server role or changes name of a user-defined server role. Fixed server roles cannot be renamed.

Syntax

```
ALTER SERVER ROLE server_role_name
{
   [ ADD MEMBER server_principal ]
  | [ DROP MEMBER server_principal ]
  | [ WITH NAME = new_server_role_name ]
} [ ; ]
```

Arguments

- Server_role_name Is the name of the server role to be changed.
- ADD MEMBER server_principal adds the specified server principal to the server role. Server_principal can be a login or a user-defined server role. Server_principal cannot be a fixed server role, a database role, or SA.
- DROP MEMBER server_principal removes the specified server principal from the server role. Server_principal can be a login or a user-defined server role. Server_principal cannot be a fixed server role, a database role, or SA.
- WITH NAME =new_server_role_nameSpecifies the new name of the user-defined server role. This name cannot already exist in the server.

Permissions

Requires ALTER ANY SERVER ROLE permission on the server to change the name of a user-defined server role.

28.3.3 Fixed server roles

To add a member to a fixed server role, you must be a member of that fixed server role, or be a member of the sysadmin fixed server role.

28.3.4 User-defined server roles

To add a member to a user-defined server role, you must be a member of the sysadmin fixed server role or have CONTROL SERVER or ALTER ANY SERVER ROLE permission. Or you must have ALTER permission on that role.

Examples

A. Changing the name of a server role

The following example creates a server role named Product, and then changes the name of server role to Production.

```
CREATE SERVER ROLE Product ;
ALTER SERVER ROLE Product WITH NAME = Production ;
GO
```

B. Adding a domain account to a server role

The following example adds a domain account named adventure-works\roberto0 to the user-defined server role named Production.

```
ALTER SERVER ROLE Production ADD MEMBER [adventure-works\roberto0] ;
```

C. Adding a SQL Server login to a server role

The following example adds a SQL Server login named Ted to the diskadmin fixed server role.

```
ALTER SERVER ROLE diskadmin ADD MEMBER Ted ;
GO
```

D. Removing a domain account from a server role

The following example removes a domain account named adventure-works\roberto0 from the user-defined server role named Production.

```
ALTER SERVER ROLE Production DROP MEMBER [adventure-works\roberto0] ;
```

E. Removing a SQL Server login from a server role

The following example removes the SQL Server login Ted from the diskadmin fixed server role.

```
ALTER SERVER ROLE Production DROP MEMBER Ted ;
GO
```

F. Granting a login the permission to add logins to a user-defined server role

The following example allows Ted to add other logins to the user-defined server role named Production.

```
GRANT ALTER ON SERVER ROLE::Production TO Ted ;
GO
```

G. To view role membership

To view role membership, use the **Server Role (Members)** page in SQL Server Management Studio or execute the following query:

```
SELECT SRM.role_principal_id, SP.name AS Role_Name,
SRM.member_principal_id, SP2.name AS Member_Name
FROM sys.server_role_members AS SRM
JOIN sys.server_principals AS SP
   ON SRM.role_principal_id = SP.principal_id
JOIN sys.server_principals AS SP2
   ON SRM.member_principal_id = SP2.principal_id
ORDER BY SP.name, SP2.name
```

28.4 DROP SERVER ROLE (Transact-SQL)

Removes a user-defined server role.

User-defined server roles are new in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3).

Syntax

DROP SERVER ROLE role_name

Arguments

Role_name Specifies the user-defined server role to be dropped from the server.

Permissions

Requires CONTROL permission on the server role or ALTER ANY SERVER ROLE permission.

Examples

A. To drop a server role

The following example drops the server role purchasing.

```
DROP SERVER ROLE purchasing;
```

GO

B. To view role membership

To view role membership, use the **Server Role (Members**) page in SQL Server Management Studio or execute the following query:

```
SELECT SRM.role_principal_id, SP.name AS Role_Name,
SRM.member_principal_id, SP2.name AS Member_Name
FROM sys.server_role_members AS SRM
JOIN sys.server_principals AS SP
   ON SRM.role_principal_id = SP.principal_id
JOIN sys.server_principals AS SP2
   ON SRM.member_principal_id = SP2.principal_id
ORDER BY SP.name, SP2.name
```

C. To view role membership

To determine whether a server role owns another server role, execute the following query:

```
SELECT SP1.name AS RoleOwner, SP2.name AS Server_Role
FROM sys.server_principals AS SP1
JOIN sys.server_principals AS SP2
   ON SP1.principal_id = SP2.owning_principal_id
ORDER BY SP1.name;
```

- Sp_addsrvrolemember and sp_dropsrvrolemember are deprecated. Use ALTER SERVER ROLE instead.
- ALTER ROLE is modified to add or remove members from roles.

28.5 ALTER ROLE (Transact-SQL)

Changes the name of a database role.

Syntax

```
ALTER ROLE role_name WITH NAME = new_name
```

Arguments

Role_name is the name of the role to be changed.
WITH NAME =new_name Specifies the new name of the role. This name must not already exist in the database.

Permissions

Requires ALTER ANY ROLE permission on the database.

Examples

The following example changes the name of role buyers to purchasing.

```
USE AdventureWorks2008R2;
ALTER ROLE buyers WITH NAME = purchasing;
GO
```

- Sp_addrolemember and sp_droprolemember are deprecated. Use ALTER ROLE instead.
- IS_ROLEMEMBER is added to check the membership of database roles.

28.6 IS_ROLEMEMBER (Transact-SQL)

Indicates whether a specified database principle is a member of the specified database role.

Syntax

IS_ROLEMEMBER ('role' [, 'database_principal'])

Arguments

- **'Role '** is the name of the database role that is being checked. *role* is sysname.
- **'database**_*principal* **'** is the name of the database user, database role, or application role to check. *Database_principal* is sysname, with a default of NULL. If no value is specified, the result is based on the current execution context. If the parameter contains the word NULL will return NULL.

Permissions

Requires VIEW DEFINITION permission on the database role.

28.7 SQL Server Audit Enhancements

Support for server auditing is expanded to include all editions of SQL Server. Database audits are limited to Enterprise, Datacenter, Developer, and Evaluation editions.

SQL Server Audit is now more resilient to failures to write to the audit log. For example, if the target directory is on a remote share and the network goes down, SQL Server Audit will now be able to recover once the network connection is re-established. In addition, a new option has been introduced to fail an operation that would otherwise generate an audit event to be written to a failed audit target. For more information, see the FAIL_OPERATION option for the ON_FAILURE event in CREATE SERVER AUDIT.

In the case of a failure during audit initiation, the server will not start. In this case, the server can be started by using the **–f** option at the command line.

When an audit failure causes the server to shut down or not to start because ON_FAILURE=SHUTDOWN is specified for the audit, the MSG_AUDIT_FORCED_SHUTDOWN event will be written to the log. Because the shutdown will occur on the first encounter of this setting, the event will be written one time. This event is written after the failure message for the audit causing the shutdown. An administrator can bypass audit-induced shutdowns by starting SQL Server in Single User mode using the **–m** flag. If you start in Single User mode, you will downgrade any audit where ON_FAILURE=SHUTDOWN is specified to run in that session as ON_FAILURE=CONTINUE. When SQL Server is started by using the **–m** flag, the MSG_AUDIT_SHUTDOWN_BYPASSED message will be written to the error log.

Previously, Audit logs could have an indeterminate number of log files or else be rolled-over after a predefined number. A new option has been introduced to cap the number of audit files without rolling over, in order to allow customers to control the amount of audit information collected without losing audit records. For more information, see the MAX_FILES option in CREATE SERVER AUDIT.

When possible, the audit log provides additional Transact-SQL stack frame information. In many cases, auditors can now determine whether a query was issued through a stored procedure or directly by an application.

SQL Server audit specifications now support a user-defined audit group. Audited events can be written to the audit log by using the new sp_audit_write (Transact-SQL) procedure. User-defined audit events allow applications to write custom information to the audit log, such as the name of the application user who has connected in cases where a common login is used to connect to SQL Server.

New columns are added to sys.server_file_audits, sys.server_audits, and sys.fn_get_audit_file to track user-defined audit events.

SQL Server Audit now supports the ability to filter audit events before they are written to the audit log. For more information, see the WHERE clause in CREATE SERVER AUDIT and ALTER SERVER AUDIT.

New audit groups support the monitoring of contained database users.

The new audit options have been added to the audit dialog boxes in Management Studio.

28.7.1 CREATE SERVER AUDIT (Transact-SQL)

Creates a server audit object using SQL Server Audit.

Syntax

```
CREATE SERVER AUDIT audit_name
{
   TO { [ FILE (<file_options> [ , ...n ] ) ] | APPLICATION_LOG | SECURITY_LOG }
   [ WITH ( <audit_options> [ , ...n ] ) ]
   [ WHERE <predicate_expression> ]
}
[ ; ]

<file_options>::=
{
      FILEPATH = 'os_file_path'
   [ , MAXSIZE = { max_size { MB | GB | TB } | UNLIMITED } ]
   [ , { MAX_ROLLOVER_FILES = { integer | UNLIMITED } } | { MAX_FILES = integer } ]
   [ , RESERVE_DISK_SPACE = { ON | OFF } ]
}

<audit_options>::=
{
   [  QUEUE_DELAY = integer ]
   [ , ON_FAILURE = { CONTINUE | SHUTDOWN | FAIL_OPERATION } ]
   [ , AUDIT_GUID = uniqueidentifier ]
}

<predicate_expression>::=
{
   [NOT ] <predicate_factor>
   [ { AND | OR } [NOT ] { <predicate_factor> } ]
   [,...n ]
}

<predicate_factor>::=
   event_field_name { = | < > | ! = | > | > = | < | < = } { number | ' string ' }
```

Arguments

TO { FILE | APPLICATION_LOG | SECURITY }

Determines the location of the audit target. The options are a binary file, The Windows Application log, or the Windows Security log. SQL Server cannot write to the Windows Security log without configuring additional settings in Windows. For more information, see Write SQL Server Audit Events to the Security Log.

FILEPATH ='*os_file_path*'

The path of the audit log. The file name is generated based on the audit name and audit GUID.

MAXSIZE = { *max_size* }

Specifies the maximum size to which the audit file can grow. The *max_size* value must be an integer followed by MB, GB, TB, or UNLIMITED. The minimum size that you can specify for *max_size* is 2 MB and the maximum is 2,147,483,647 TB. When UNLIMITED is specified, the file grows until the disk is full. Specifying a value lower than 2 MB will raise the error MSG_MAXSIZE_TOO_SMALL. The default value is UNLIMITED.

MAX_ROLLOVER_FILES ={ *integer* | UNLIMITED }

Specifies the maximum number of files to retain in the file system in addition to the current file. The *MAX_ROLLOVER_FILES* value must be an integer or UNLIMITED. The default value is UNLIMITED. This parameter is evaluated whenever the audit restarts (which can happen when the instance of the Database Engine restarts or when the audit is turned off and then on again) or when a new file is needed because the MAXSIZE has been reached. When *MAX_ROLLOVER_FILES* is evaluated, if the number of files exceeds the *MAX_ROLLOVER_FILES* setting, the oldest file is deleted. As a result, when the setting of *MAX_ROLLOVER_FILES* is 0 a new file is created each time the *MAX_ROLLOVER_FILES* setting is evaluated. Only one file is automatically deleted when *MAX_ROLLOVER_FILES* setting is evaluated, so when the value of *MAX_ROLLOVER_FILES* is decreased, the number of files will not shrink unless old files are manually deleted. The maximum number of files that can be specified is 2,147,483,647.

MAX_FILES =*integer*

Specifies the maximum number of audit files that can be created. Does not rollover to the first file when the limit is reached. When the MAX_FILES limit is reached, any action that causes additional audit events to be generated will fail with an error.

RESERVE_DISK_SPACE = { ON | OFF }

This option pre-allocates the file on the disk to the MAXSIZE value. It applies only if MAXSIZE is not equal to UNLIMITED. The default value is OFF.

QUEUE_DELAY =*integer*

Determines the time, in milliseconds, that can elapse before audit actions are forced to be processed. A value of 0 indicates synchronous delivery. The minimum settable query delay value is 1000 (1 second), which is the default. The maximum is 2,147,483,647 (2,147,483.647 seconds or 24 days, 20 hours, 31 minutes, 23.647 seconds). Specifying an invalid number will raise the error MSG_INVALID_QUEUE_DELAY.

ON_FAILURE = { CONTINUE | SHUTDOWN | FAIL_OPERATION }

Indicates whether the instance writing to the target should fail, continue, or stop SQL Server if the target cannot write to the audit log. The default value is CONTINUE.

CONTINUE

SQL Server operations continue. Audit records are not retained. The audit continues to attempt to log events and will resume if the failure condition is resolved. Selecting the continue option can allow unaudited activity which could violate your security policies. Use this option, when continuing operation of the Database Engine is more important than maintaining a complete audit.

SHUTDOWN

Forces a server shut down when the server instance writing to the target cannot write data to the audit target. The login issuing this must have the SHUTDOWN permission. If the logon does not have this permission, this function will fail and an error message will be raised. No audited events occur. Use the option when an audit failure could compromise the security or integrity of the system.

FAIL_OPERATION

Database actions fail if they cause audited events. Actions which do not cause audited events can continue, but no audited events can occur. The audit continues to attempt to log events and will resume if the failure condition is resolved. Use this option when maintaining a complete audit is more important than full access to the Database Engine.

AUDIT_GUID =*uniqueidentifier*

To support scenarios such as database mirroring, an audit needs a specific GUID that matches the GUID found in the mirrored database. The GUID cannot be modified after the audit has been created.

predicate_expression

Specifies the predicate expression used to determine if an event should be processed or not. Predicate expressions are limited to 3000 characters, which limits string arguments.

event_field_name

Is the name of the event field that identifies the predicate source. Audit fields are described in sys.fn_get_audit_file (Transact-SQL). All fields can be audited except file_name and audit_file_offset.

Number

Is any numeric type including decimal. Limitations are the lack of available physical memory or a number that is too large to be represented as a 64-bit integer.

String

Either an ANSI or Unicode string as required by the predicate compare. No implicit string type conversion is performed for the predicate compare functions. Passing the wrong type results in an error.

Permissions

- To create, alter, or drop a server audit, principals require the ALTER ANY SERVER AUDIT or the CONTROL SERVER permission.
- When you are saving audit information to a file, to help prevent tampering, restrict access to the file location.

28.7.2 ALTER SERVER AUDIT (Transact-SQL)

Alters a server audit object using the SQL Server Audit feature.

Syntax

```
ALTER SERVER AUDIT audit_name
{
   [ TO { { FILE ( <file_options> [, ...n] ) } | APPLICATION_LOG | SECURITY_LOG } ]
   [ WITH ( <audit_options> [ , ...n] ) ]
   [ WHERE <predicate_expression> ]
}
```

| REMOVE WHERE
| MODIFY NAME = new_audit_name
[;]

<file_options>::=
{
FILEPATH = 'os_file_path'
| MAXSIZE = { max_size { MB | GB | TB } | UNLIMITED }
| MAX_ROLLOVER_FILES = { integer | UNLIMITED }
| MAX_FILES = integer
| RESERVE_DISK_SPACE = { ON | OFF }
}

<audit_options>::=
{
QUEUE_DELAY = integer
| ON_FAILURE = { CONTINUE | SHUTDOWN | FAIL_OPERATION }
| STATE = = { ON | OFF }
}

<predicate_expression>::=
{
[NOT] <predicate_factor>
[{ AND | OR } [NOT] { <predicate_factor> }]
[,...n]
}

<predicate_factor>::=
event_field_name { = | < > | ! = | > | > = | < | < = } { number | ' string ' }

Arguments

TO { FILE | APPLICATION_LOG | SECURITY }

Determines the location of the audit target. The options are a binary file, the Windows application log, or the Windows security log.

FILEPATH = '*os_file_path*'

The path of the audit trail. The file name is generated based on the audit name and audit GUID.

MAXSIZE =*max_size*

Specifies the maximum size to which the audit file can grow. The *max_size* value must be an integer followed by **MB**, **GB**, **TB**, or **UNLIMITED**. The minimum size that you can specify for *max_size* is 2 **MB** and the maximum is 2,147,483,647 **TB**. When **UNLIMITED** is specified the file grows until the disk is full. Specifying a value lower than 2 MB will raise the error MSG_MAXSIZE_TOO_SMALL. The default value is **UNLIMITED**.

MAX_ROLLOVER_FILES **=***integer* | **UNLIMITED**

Specifies the maximum number of files to retain in the file system. When the setting of MAX_ROLLOVER_FILES=0 there is no limit imposed on the number of rollover files that will be created. The default value is 0. The maximum number of files that can be specified is 2,147,483,647.

MAX_FILES =*integer*

Specifies the maximum number of audit files that can be created. Does not rollover to the first file when the limit is reached. When the MAX_FILES limit is reached, any action that causes additional audit events to be generated will fail with an error.

RESERVE_DISK_SPACE **=** { ON | OFF }

This option pre-allocates the file on the disk to the MAXSIZE value. Only applies if MAXSIZE is not equal to UNLIMITED. The default value is OFF.

QUEUE_DELAY **=***integer*

Determines the time in milliseconds that can elapse before audit actions are forced to be processed. A value of 0 indicates synchronous delivery. The minimum settable query delay value is 1000 (1 second), which is the default. The maximum is 2,147,483,647 (2,147,483.647 seconds or 24 days, 20 hours, 31 minutes, 23.647 seconds). Specifying an invalid number will raise the error MSG_INVALID_QUEUE_DELAY.

ON_FAILURE **=** { CONTINUE | SHUTDOWN | FAIL_OPERATION}

Indicates whether the instance writing to the target should fail, continue, or stop if SQL Server cannot write to the audit log.

CONTINUE

SQL Server operations continue. Audit records are not retained. The audit continues to attempt to log events and will resume if the failure condition is resolved. Selecting the continue option can allow unaudited activity which could violate your security policies. Use this option, when continuing operation of the Database Engine is more important than maintaining a complete audit.

SHUTDOWN

Forces a server shut down when the server instance writing to the target cannot write data to the audit target. The login issuing this must have the SHUTDOWN permission. If the logon does not have this permission, this function will fail and an error message will be raised. No audited events occur. Use the option when an audit failure could compromise the security or integrity of the system.

FAIL_OPERATION

Database actions fail if they cause audited events. Actions which do not cause audited events can continue, but no audited events can occur. The audit continues to attempt to log events and will resume if the failure condition is resolved. Use this option when maintaining a complete audit is more important than full access to the Database Engine.

STATE = { ON | OFF }

Enables or disables the audit from collecting records. Changing the state of a running audit (from ON to OFF) creates an audit entry that the audit was stopped, the principal that stopped the audit, and the time the audit was stopped.

MODIFY NAME = *new_audit_name*

Changes the name of the audit. Cannot be used with any other option.

predicate_expression

Specifies the predicate expression used to determine if an event should be processed or not. Predicate expressions are limited to 3000 characters, which limits string arguments.

event_field_name

Is the name of the event field that identifies the predicate source. Audit fields are described in sys.fn_get_audit_file (Transact-SQL). All fields can be audited except file_name and audit_file_offset.

Number

Is any numeric type including decimal. Limitations are the lack of available physical memory or a number that is too large to be represented as a 64-bit integer.

String

Either an ANSI or Unicode string as required by the predicate compare. No implicit string type conversion is performed for the predicate compare functions. Passing the wrong type results in an error.

Permissions

To create, alter, or drop a server audit principal, you must have ALTER ANY SERVER AUDIT or the CONTROL SERVER permission.

29 Beyond Relational

In the area of providing a rich experience over unstructured data we have:

- Lots of performance and scale work in Full-Text Search!
- Customizable NEAR in FTS
- The ability to search only within document properties instead of the full document
- Semantic Similarity Search between documents. This provides you the ability to answer questions such as: "Find documents that talk about the same thing as this other document!"
- Better scalability and performance for FileStream data, including the ability to store the data in multiple containers
- Full Win 32 application compatibility for unstructured data stored in a new table called FILETABLE. You create a Filetable and can drag and drop your documents into the database and run your favorite Windows applications on them (e.g., Office, Windows Explorer).

In the spatial area we have lot of additional items including:

- Circular arc support
- GEOMETRY and GEOGRAPHY method parity
- FullGlobe support for GEOGRAPHY
- Spatial aggregation
- Performance improvements
- Better predictable default index
- Index support for nearest neighbor queries

30 Metadata Discovery

The metadata discovery improvement in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) allows SQL Server Native Client applications to ensure that column or parameter metadata returned from the execution of a query is identical to or compatible with the metadata format specified before executing the query. An error is received if the metadata returned after query execution is not compatible with the metadata format you specified before query execution.

Bcp and ODBC functions, and IBCPSession and IBCPSession2 interfaces, can now specify a delayed read (delayed metadata discovery) to avoid metadata discovery for query out operations. This improves performance and eliminates metadata discovery failures.

The following bcp functions have been enhanced in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) to provide improved metadata discovery:

30.1 bcp functions

30.1.1 bcp_columns

Sets the total number of columns found in the user file for use with a bulk copy into or out of SQL Server.bcp_setbulkmode can be used instead of **bcp_columns** and bcp_colfmt.

Syntax

```
RETCODE bcp_columns (
    HDBC hdbc,
    INT nColumns);
```

Arguments

hdbc

Is the bulk copy-enabled ODBC connection handle.

nColumns

Is the total number of columns in the user file. Even if you are preparing to bulk copy data from the user file to an SQL Server table and do not intend to copy all columns in the user file, you must still set ***nColumns*** to the total number of user-file columns

30.1.2 bcp_control

Changes the default settings for various control parameters for a bulk copy between a file and SQL Server.

Syntax

RETCODE bcp_control (

HDBC hdbc,

INT eOption,

void iValue);*

Arguments

hdbc

Is the bulk copy-enabled ODBC connection handle.

eOption

Is one of the following:

30.1.3 BCPABORT

Stops a bulk-copy operation that is already in progress. Call **bcp_control** with an ***eOption*** of BCPABORT from another thread to stop a running bulk copy operation. The ***iValue*** parameter is ignored.

30.1.4 BCPBATCH

Is the number of rows per batch. The default is 0, which indicates either all rows in a table, when data is being extracted, or all rows in the user data file, when data is being copied to an SQL Server. A value less than 1 resets BCPBATCH to the default.

30.1.5 BCPDELAYREADFMT

A Boolean, if set to true, will cause bcp_readfmt to read at execution. If false (the default), **bcp_readfmt** will immediately read the format file. A sequence error will occur if BCPDELAYREADFMT is true and you call**bcp_columns** or **bcp_setcolfmt**.

A sequence error will also occur if you call bcp_control(hdbc, BCPDELAYREADFMT, (void *)FALSE) after callingbcp_control(hdbc, BCPDELAYREADFMT, (void *)TRUE) and **bcp_writefmt**.

For more information, see Metadata Discovery.

30.1.6 BCPFILECP

iValue contains the number of the code page for the data file. You can specify the number of the code page, such as 1252 or 850, or one of these values:

BCPFILE_ACP: data in the file is in the Microsoft Windows® code page of the client.

BCPFILE_OEMCP: data in the file is in the OEM code page of the client (default).

BCPFILE_RAW: data in the file is in the code page of the SQL Server.

30.1.7 BCPFILEFMT

The version number of the data file format. This can be 80 (SQL Server 2000), 90 (SQL Server 2005), 100 (SQL Server 2008 or SQL Server 2008 R2), or 110 (Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3)). 110 is the default. This is useful for exporting and importing data in formats that were supported by earlier version of the server. For example, to import data that was obtained from a text column in a SQL Server 2000 server into a **varchar(max)** column in a SQL Server 2005 or later server, you should specify 80. Similarly, if you specify 80 when exporting data from a **varchar(max)** column, it will be saved just like text columns are saved in the SQL Server 2000 format, and can be imported into a text column of a SQL Server 2000 server.

30.1.8 BCPFIRST

Is the first row of data to file or table to copy. The default is 1; a value less than 1 resets this option to its default.

30.1.9 BCPFIRSTEX

For BCP out operations, specifies the first row of the database table to copy into the data file.

For BCP in operations, specifies the first row of the data file to copy into the database table.

The *iValue* parameter is expected to be the address of a signed 64-bit integer containing the value. The maximum value that can be passed to BCPFIRSTEX is 2^63-1.

30.1.10 BCPFMTXML

Specifies that the format file generated should be in XML format. It is off by default.

XML format files provide greater flexibility but with some added constraints. For example, you can not specify the prefix and terminator for a field simultaneously, which was possible in older format files.

30.1.11 BCPHINTS

iValue contains an SQLTCHAR character string pointer. The string addressed specifies either SQL Server bulk-copy processing hints or a Transact-SQL statement that returns a result set. If a Transact-SQL statement is specified that returns more than one result set, all result sets after the first are ignored. For more information about bulk-copy processing hints, see bcp Utility.

30.1.12 BCPKEEPIDENTITY

When *iValue* is TRUE, specifies that bulk copy functions insert data values supplied for SQL Server columns defined with an identity constraint. The input file must supply values for the identity columns. If this is not set, new identity values are generated for the inserted rows. Any data present in the file for the identity columns is ignored.

30.1.13 BCPKEEPNULLS

Specifies whether empty data values in the file will be converted to NULL values in the SQL Server table. When*iValue* is TRUE, empty values will be converted to NULL in the SQL Server table. The default is for empty values to be converted to a default value for the column in the SQL Server table if a default exists.

30.1.14 BCPLAST

Is the last row to copy. The default is to copy all rows; a value less than 1 resets this option to its default.

30.1.15 BCPLASTEX

For BCP out operations, specifies the last row of the database table to copy into the data file.

For BCP in operations, specifies the last row of the data file to copy into the database table.

The *iValue* parameter is expected to be the address of a signed 64-bit integer containing the value. The maximum value that can be passed to BCPLASTEX is 2^63-1.

30.1.16 BCPMAXERRS

Is the number of errors allowed before the bulk copy operation fails. The default is 10; a value less than 1 resets this option to its default. Bulk copy imposes a maximum of 65,535 errors. An attempt to set this option to a value larger than 65,535 results in the option being set to 65,535.

30.1.17 BCPODBC

When TRUE, specifies that datetime and smalldatetime values saved in character format will use the ODBC timestamp escape sequence prefix and suffix. The BCPODBC option only applies to BCP_OUT.

When FALSE, a datetime value representing January 1, 1997 is converted to the character string: 1997-01-01 00:00:00.000. When TRUE, the same datetime value is represented as: {ts '1997-01-01 00:00:00.000'}.

30.1.18 BCPROWCOUNT

Returns the number of rows affected by the current (or last) BCP operation.

30.1.19 BCPTEXTFILE

When TRUE, specifies that the data file is a text file, rather than a binary file. If the file is a text file, BCP determines whether or not it is Unicode by checking the Unicode byte marker in the first two bytes of the data file.

30.1.20 BCPUNICODEFILE

When TRUE, specifies the input file is a Unicode file.

iValue

Is the value for the specified *eOption*. *iValue* is an integer (LONGLONG) value cast to a void pointer to allow for future expansion to 64 bit values.

30.1.21 bcp_getcolfmt

Used to find the column format property value.

Syntax

RETCODE bcp_getcolfmt (

HDBC hdbc,

INT field,

INT property,

void pValue,*

INT cbvalue,

INT pcbLen);*

Arguments

hdbc

Is the bulk copy-enabled ODBC connection handle.

field

Is the column number for which the property is retrieved.

property

Is one of the property constants.

pValue

Is the pointer to the buffer from which to retrieve the property value.

cbValue

Is the length of the property buffer in bytes.

pcbLen

Pointer to length of the data that is being returned in the property buffer.

30.1.22 bcp_readfmt

Reads a data file format definition from the specified format file.

Syntax

RETCODE bcp_readfmt (

HDBC hdbc,

LPCTSTR szFormatFile);

Arguments

hdbc

Is the bulk copy-enabled ODBC connection handle.

szFormatFile

Is the path and file name of the file containing the format values for the data file.

30.1.23 bcp_setcolfmt

The bcp_setcolfmt function supersedes the bcp_colfmt. In specifying the column collation, the bcp_setcolfmt function must be used. bcp_setbulkmode can be used to specify more than one column format.

This function provides a flexible approach to specifying the column format in a bulk copy operation. It is used to set individual column format attributes. Each call to bcp_setcolfmt sets one column format attribute.

The bcp_setcolfmt function specifies the source or target format of the data in a user file. When used as a source format, bcp_setcolfmt specifies the format of an existing data file used as a data source of data in a bulk copy to a table in SQL Server. When used as a target format, the data file is created using the column formats specified with bcp_setcolfmt.

Syntax

RETCODE bcp_setcolfmt (

HDBC hdbc,

INT field,

INT property,

void pValue,*

INT cbValue);

Arguments

hdbc

Is the bulk copy-enabled ODBC connection handle.

field

Is the ordinal column number for which the property is being set.

property

Is one of the property constants. Property constants are defined in this table.

Property	Value	Description
BCP_FMT_TYPE	BYTE	Is the data type of this column in the user file. If different from the data type of the corresponding column in the database table, bulk copy converts the data if possible. The BCP_FMT_TYPE parameter is enumerated by the SQL Server data type tokens in sqlncli.h,

		rather than the ODBC C data type enumerators. For example, you can specify a character string, ODBC type SQL_C_CHAR, using the SQLCHARACTER type specific to SQL Server. To specify the default data representation for the SQL Server data type, set this parameter to 0. For a bulk copy out of SQL Server into a file, when BCP_FMT_TYPE is SQLDECIMAL or SQLNUMERIC: • If the source column is not **decimal** or **numeric**, the default precision and scale are used. • If the source column is **decimal** or **numeric**, the precision and scale of the source column are used.
BCP_FMT_INDICATOR_LEN	INT	Is the length in bytes of the indicator (prefix). It is the length, in bytes, of a length/null indicator within the column data. Valid indicator length values are 0 (when using no indicator), 1, 2, or 4. To specify default bulk copy indicator usage, set this parameter to SQL_VARLEN_DATA. Indicators appear in memory directly before any data, and in the data file directly before the data to which they apply. If more than one means of specifying a data file column length is used (such as an indicator and a maximum column length, or an indicator and a terminator sequence), bulk copy chooses the one that results in the least amount of data being copied. Data files generated by bulk copy when no user intervention adjusts the format of the data contain indicators when the column data can vary in length or the column can accept NULL as a value.

BCP_FMT_DATA_LEN	DBINT	Is the length in bytes of the data (column length) It is the maximum length, in bytes, of this column's data in the user file, not including the length of any length indicator or terminator. Setting BCP_FMT_DATA_LEN to SQL_NULL_DATA indicates that all values in the data file column are, or should be set to, NULL. Setting BCP_FMT_DATA_LEN to SQL_VARLEN_DATA indicates that the system should determine the length of data in each column. For some columns, this could mean that a length/null indicator is generated to precede data on a copy from SQL Server, or that the indicator is expected in data copied to SQL Server. For SQL Server character and binary data types, BCP_FMT_DATA_LEN can be SQL_VARLEN_DATA, SQL_NULL_DATA, 0, or some positive value. If BCP_FMT_DATA_LEN is SQL_VARLEN_DATA, the system uses either the length indicator, if present, or a terminator sequence to determine the length of the data. If both a length indicator and a terminator sequence are supplied, bulk copy uses the one that results in the least amount of data being copied. If BCP_FMT_DATA_LEN is SQL_VARLEN_DATA, the data type is an SQL Server character or binary type, and neither a length indicator nor a terminator sequence is specified, the system returns an error message. If BCP_FMT_DATA_LEN is 0 or a positive value, the system uses BCP_FMT_DATA_LEN as the maximum data length. However, if, in addition to a positive BCP_FMT_DATA_LEN, a length indicator or terminator sequence is provided, the system determines the data length by using the method that results in the least amount of data being copied. The BCP_FMT_DATA_LEN value represents the

		count of bytes of data. If character data is represented by Unicode wide characters, then a positive BCP_FMT_DATA_LEN parameter value represents the number of characters multiplied by the size, in bytes, of each character.
BCP_FMT_TERMINATOR	LPCBYTE	Pointer to the terminator sequence (either ANSI or Unicode as appropriate) to be used for this column. This parameter is useful mainly for character data types because all other types are of fixed length or, in the case of binary data, require an indicator of length to accurately record the number of bytes present. To avoid terminating extracted data, or to indicate that data in a user file is not terminated, set this parameter to NULL. If more than one means of specifying a user-file column length is used (such as a terminator and a length indicator, or a terminator and a maximum column length), bulk copy chooses the one that results in the least amount of data being copied. The bulk copy API performs Unicode-to-MBCS character conversion as required. Care must be taken to ensure that both the terminator byte string and the length of the byte string are set correctly.
BCP_FMT_SERVER_COL	INT	Ordinal position of the column in the database
BCP_FMT_COLLATION	LPCSTR	Collation name.

pValue

Is the pointer to the value to associate to the ***property***. It allows each column format property to be set individually.

cbvalue

Is the length of the property buffer in bytes.

Performance is improved by specifying metadata format using bcp_setbulkmode.

Bcp_control has a new *eOption* to control the behavior of bcp_readfmt**:** BCPDELAYREADFMT.

The following ODBC functions have been enhanced in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) to provide improved metadata discovery:

30.2 ODBC functions

30.2.1 SQLNumResultCols

For executed statements, the SQL Server Native Client ODBC driver does not visit the server to report the number of columns in a result set. In this case, SQLNumResultCols does not cause a server roundtrip. Like SQLDescribeCol andSQLColAttribute, calling SQLNumResultCols on prepared but not executed statements generates a server roundtrip.

When a Transact-SQL statement or statement batch returns multiple result row sets, it is possible for the number of result set columns to change from one set to another. SQLNumResultCols should be called for each set. When the number of columns changes, the application should rebind data values prior to fetching row results.

Improvements in the database engine starting with Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) allow SQLNumResultCols to obtain more accurate descriptions of the expected results. These more accurate results may differ from the values returned by **S**QLNumResultCols in previous versions of SQL Server.

30.2.2 SQLDescribeCol

For executed statements, the SQL Server Native Client ODBC driver does not need to query the server to describe columns in a result set. In this case, SQLDescribeCol does not cause a server roundtrip. LikeSQLColAttribute and SQLNumResultCols, calling SQLDescribeCol on prepared but not executed statements generates a server roundtrip.

When a Transact-SQL statement or statement batch returns multiple result row sets, it is possible for a column, referenced by ordinal, to originate in a separate table or to refer to an entirely different column in the result set.SQLDescribeCol should be called for each set. When the result set changes, the

application should rebind data values prior to fetching row results. For more information about handling multiple result set returns, see SQLMoreResults.

Column attributes are reported for only the first result set when multiple result sets are generated by a prepared batch of SQL statements.

For large value data types, the value returned in ***DataTypePtr*** is SQL_VARCHAR, SQL_VARBINARY, or SQL_NVARCHAR. A value of SQL_SS_LENGTH_UNLIMITED in ***ColumnSizePtr*** indicates that the size is "unlimited".

Improvements in the database engine starting with Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3) allow **S**QLDescribeCol to obtain more accurate descriptions of the expected results. These more accurate results may differ from the values returned by SQLDescribeCol in previous versions of SQL Server.

30.2.3 SQLDescribeCol Support for Enhanced Date and Time Features

The values returned for date/time types are as follows:

	DataTypePtr	*ColumnSizePtr*	*DecimalDigitsPtr*
datetime	SQL_TYPE_TIMESTAMP	23	3
smalldatetime	SQL_TYPE_TIMESTAMP	16	0
Date	SQL_TYPE_DATE	10	0
Time	SQL_SS_TIME2	8, 10..16	0..7
datetime2	SQL_TYPE_TIMESTAMP	19, 21..27	0..7
datetimeoffset	SQL_SS_TIMESTAMPOFFSET	26, 28..34	0..7

30.2.4 SQLNumParams

Improvements in the database engine beginning with Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3) allow SQLNumParams to obtain more accurate descriptions of the expected results. These more accurate results may differ from the values returned by SQLNumParams in previous versions of SQL Server.

30.2.5 SQLDescribeParam

To describe the parameters of any SQL statement, the SQL Server Native Client ODBC driver builds and executes a Transact-SQL SELECT statement when SQLDescribeParam is called on a prepared ODBC statement handle. The metadata of the result set determines the characteristics of the parameters in the prepared statement. SQLDescribeParam can return any error code that SQLExecute or SQLExecDirect might return.

Improvements in the database engine starting with Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3) allow SQLDescribeParam to obtain more accurate descriptions of the expected results. These more accurate results may differ from the values returned by SQLDescribeParam in previous versions of SQL Server. Also new in Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), *ParameterSizePtr*now returns a value that aligns with the definition for the size, in characters, of the column or expression of the corresponding parameter marker as defined in the ODBC specification. In previous versions of SQL Server Native Client,*ParameterSizePtr* could be the corresponding value of SQL_DESC_OCTET_LENGTH for the type, or an irrelevant column size value that was supplied to SQLBindParameter for a type, the value of which should be ignored (SQL_INTEGER, for example).

The driver does not support calling SQLDescribeParam in the following situations:

- After SQLExecDirect for any Transact-SQL UPDATE or DELETE statements containing the FROM clause.
- For any ODBC or Transact-SQL statement containing a parameter in a HAVING clause, or compared to the result of a SUM function.
- For any ODBC or Transact-SQL statement depending on a subquery containing parameters.
- For ODBC SQL statements containing parameter markers in both expressions of a comparison, like, or quantified predicate.
- For any queries where one of the parameters is a parameter to a function.
- When there are comments (/* */) in the Transact-SQL command.

When processing a batch of Transact-SQL statements, the driver also does not support calling SQLDescribeParam for parameter markers in statements after the first statement in the batch.

When describing the parameters of prepared stored procedures, SQLDescribeParam uses the system stored procedure sp_sproc_columns to retrieve parameter characteristics. **s**p_sproc_columns can report data for stored procedures within the current user database. Preparing a fully qualified stored procedure name allowsSQLDescribeParam to execute across databases. For example, the system stored procedure sp_who can be prepared and executed in any database as:

```
SQLPrepare(hstmt, "{call sp_who(?)}", SQL_NTS);
```

Executing SQLDescribeParam after successful preparation returns an empty row set when connected to any database but master. The same call, prepared as follows, causes SQLDescribeParam to succeed regardless of the current user database:

SQLPrepare(hstmt, "{call master..sp_who(?)}", SQL_NTS);

For large value data types, the value returned in *DataTypePtr* is SQL_VARCHAR, SQL_VARBINARY, or SQL_NVARCHAR. To indicate that the size of the large value data type parameter is "unlimited," the SQL Server Native Client ODBC driver sets *ParameterSizePtr* to 0. Actual size values are returned for standard varchar parameters.

To bind an "unlimited" size input parameter, data-at-execution must be used. It is not possible to bind an "unlimited" size output parameter (there is no method for streaming data from an output parameter, like SQLGetData does for result sets).

For output parameters, a buffer must be bound and if the value is too large, the buffer is filled and a SQL_SUCCESS_WITH_INFO message and is returned along with the "string data; right truncation" warning. The data that was truncated is then discarded.

30.2.6 SQLDescribeParam and Table-Valued Parameters

An application can retrieve table-valued parameter information for a prepared statement with SQLDescribeParam.

30.2.7 SQLDescribeParam Support for Enhanced Date and Time Features

The values returned for date/time types are as follows:

	DataTypePtr	*ParameterSizePtr*	*DecimalDigitsPtr*
datetime	SQL_TYPE_TIMESTAMP	23	3
smalldatetime	SQL_TYPE_TIMESTAMP	16	0
Date	SQL_TYPE_DATE	10	0
Time	SQL_SS_TIME2	8, 10..16	0..7
datetime2	SQL_TYPE_TIMESTAMP	19, 21..27	0..7
datetimeoffset	SQL_SS_TIMESTAMPOFFSET	26, 28..34	0..7

The following OLE DB member functions have been enhanced in Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3) to provide improved metadata discovery:

- IColumnsInfo::GetColumnInfo
- IColumnsRowset::GetColumnsRowset
- ICommandWithParameters::GetParameterInfo

performance is improved also by specifying metadata format using IBCPSession::BCPSetBulkMode

The improved metadata discovery in SQL Server Native Client is possible because of the addition of two stored procedures in Microsoft SQL Server Code-Named "Denali", Community Technology Preview 3 (CTP 3):

- sp_describe_first_result_set
- sp_describe_undeclared_parameters

31 FILESTREAM Support

FILESTREAM provides a way to store and access large binary values, either through SQL Server or by direct access to the Windows file system. A large binary value is a value larger than 2 gigabytes (GB).

When a database connection is opened, @@TEXTSIZE will be set to -1 ("unlimited"), by default.

It is also possible to access and update FILESTREAM columns using Windows file system APIs.

31.1 FILESTREAM Support (OLE DB)

To send and receive varbinary(max) values greater than 2 GB, an application uses DBTYPE_IUNKNOWN in parameter and result bindings. For parameters the provider must call **IUnknown::QueryInterface** for **ISequentialStream** and for results that return **ISequentialStream**.

For OLE DB, checking related to **ISequentialStream** values will be relaxed.
When *wType* is DBTYPE_IUNKNOWN in theDBBINDING struct, length checking can be disabled either by omitting DBPART_LENGTH from *dwPart* or by setting the length of the data (at offset *obLength* in the data buffer) to ~0. In this case, the provider will not check the length of the value and will request and return all of the data available through the stream. This change will be applied to all large object (LOB) types and XML, but only when connected to SQL Server 2005 (or later) servers. This will provide greater flexibility for developers, while maintaining consistency and backwards compatibility for existing applications and downlevel servers.

This change affects all interfaces that transfer data, principally **IRowset::GetData**, **ICommand::Execute**, and**IRowsetFastLoad::InsertRow**.

31.2 FILESTREAM Support (ODBC)

To send and receive varbinary(max) values greater than 2 GB, an application must bind parameters by using**SQLBindParameter** with *ColumnSize* set to SQL_SS_LENGTH_UNLIMITED, and set the contents of *StrLen_or_IndPtr* toSQL_DATA_AT_EXEC before **SQLExecDirect** or **SQLExecute**.

As with any data-at-execution parameter, the data will be supplied with **SQLParamData** and **SQLPutData**.

You can call **SQLGetData** to fetch data in chunks for a FILESTREAM column if the column is not bound with **SQLBindCol**.

You can update FILESTREAM data if it is bound with **SQLBindCol**.

If you call **SQLFetch** on a bound column, you will receive a "data truncated" warning if the buffer is not large enough to hold the entire value. Ignore this warning and update the data in this bound column with **SQLParamData** and**SQLPutData** calls. You can update FILESTREAM data by using **SQLSetPos** if it is bound with **SQLBindCol**.

Example

FILESTREAM columns behave exactly like varbinary(max) columns, but without a size limit. They are bound as SQL_VARBINARY. (SQL_LONGVARBINARY is used with image columns, and there are restrictions on this type. For example, SQL_LONGVARBINARY connot be used as an output parameter.) The following examples show direct NTFS access for FILESTREAM columns. These examples assume that the following Transact-SQL code has been executed in the database:

CREATE TABLE fileStreamDocs(

id uniqueidentifier ROWGUIDCOL NOT NULL UNIQUE,

author varchar(64),

document VARBINARY(MAX) FILESTREAM NULL)

31.3 Access FILESTREAM Data with OpenSqlFilestream

The **OpenSqlFilestream** API obtains a Win32 compatible file handle for a FILESTREAM binary large object (BLOB) that is stored in the file system. The handle can be passed to any of the following Win32 APIs: ReadFile, WriteFile, TransmitFile,SetFilePointer, SetEndOfFile, or FlushFileBuffers. If you pass this handle to any other Win32 API, the error ERROR_ACCESS_DENIED is returned. The handle must be closed by passing it to the Win32 CloseHandle API before the transaction is committed or rolled back. Failing to close the handle will cause server-side resource leaks.

All FILESTREAM data container access must be performed in a SQL Server transaction. Transact-SQL statements can also be executed in the same transaction. This maintains consistency between the SQL data and FILESTREAM BLOB data.

To access the FILESTREAM BLOB by using Win32, Windows Authorization must be enabled.

Syntax

HANDLE OpenSqlFilestream (

LPCWSTR FilestreamPath,

SQL_FILESTREAM_DESIRED_ACCESS DesiredAccess,

ULONG OpenOptions,

LPBYTE FilestreamTransactionContext,

SIZE_T FilestreamTransactionContextLength,

PLARGE_INTEGER AllocationSize);

Parameters

FilestreamPath

[in] Is the nvarchar(max) path that is returned by the PathName function. PathName must be called from the context of an account that has SQL Server SELECT or UPDATE permissions on the FILESTREAM table and column.

DesiredAccess

[in] Sets the mode used to access FILESTREAM BLOB data. This value is passed to the DeviceIoControl Function.

OpenOptions

[in] The file attributes and flags. This parameter can also include any combination of the following flags.

Flag	Value	Meaning
SQL_FILESTREAM_OPEN_NONE	0x00000000:	The file is being opened or created with no special options.

SQL_FILESTREAM_OPEN_FLAG_ASYNC	0x00000001L	The file is being opened or created for asynchronous I/O.
SQL_FILESTREAM_OPEN_FLAG_NO_BUFFERING	0x00000002L	The system opens the file by using no system caching.
SQL_FILESTREAM_OPEN_FLAG_NO_WRITE_THROUGH	0x00000004L	The system does not write through an intermediate cache. Writes go directly to disk.
SQL_FILESTREAM_OPEN_FLAG_SEQUENTIAL_SCAN	0x00000008L	A file is accessed sequentially from beginning to end. The system can use this as a hint to optimize file caching. If an application moves the file pointer for random access, optimal caching may not occur.
SQL_FILESTREAM_OPEN_FLAG_RANDOM_ACCESS	0x00000010L	A file is accessed randomly.

		The system can use this as a hint to optimize file caching.

FilestreamTransactionContext

[in] The value that is returned by the GET_FILESTREAM_TRANSACTION_CONTEXT function.

FilestreamTransactionContextLength

[in] Number of bytes in the varbinary(max) data that is returned by the GET_FILESTREAM_TRANSACTION_CONTEXT function. The function returns an array of N bytes. N is determined by the function and is a property of the byte array that is returned.

AllocationSize

[in] Specifies the initial allocation size of the data file in bytes. It is ignored in read mode. This parameter can be NULL, in which case the default file system behavior is used.

31.4 Querying for FILESTREAM Columns

Schema rowsets in OLE DB will not report whether a column is a FILESTREAM column. ITableDefinition in OLE DB cannot be used to create a FILESTREAM column.

Catalog functions such as SQLColumns in ODBC will not report whether a column is a FILESTREAM column.

To create FILESTREAM columns or to detect which existing columns are FILESTREAM columns, you can use theis_filestream column of the sys.columns catalog view.

The following is an example:

```
-- Create a table with a FILESTREAM column.

CREATE TABLE Bob_01 (GuidCol1 uniqueidentifier ROWGUIDCOL NOT NULL UNIQUE DEFAULT NEWID(),
IntCol2 int, varbinaryCol3 varbinary(max) FILESTREAM)


-- Find FILESTREAM columns.
```

SELECT name FROM Sys.columns where is_filestream=1

-- Determine whether a column is a FILESTREAM column.

SELECT is_filestream FROM Sys.columns where name = 'varbinaryCol3' and object_id IN (SELECT object_id from Sys.tables where name='Bob_01')

31.5 Down-Level Compatibility

If your client was compiled using the version of SQL Server Native Client that was included with SQL Server 2005, and the application connects to Microsoft SQL Server Code-Named “Denali”, Community Technology Preview 3 (CTP 3), varbinary (max) behavior will be compatible with SQL Server 2005. That is, the maximum size of returned data will be limited to 2 GB. For result values larger that 2 GB, truncation will occur and a "string data right truncation" warning will be returned.

When data-type compatibility is set to 80, client behavior will be consistent with down-level client behavior.

For clients that use SQLOLEDB or other providers that were released before the SQL Server 2005 version of SQL Server Native Client, varbinary (max) will be mapped to image.

32 File Tables

FileTable is used for applications that require file and directory storage in the database, with Windows API compatibility and non-transactional access. A FileTable is a specialized user table with a pre-defined schema that stores FILESTREAM data, as well as file and directory hierarchy information and file attributes.

A FileTable provides the following functionality:

- A FileTable represents a hierarchy of directories and files. It stores data related to all the nodes in that hierarchy, for both directories and the files they contain. This hierarchy starts from a root directory that you specify when you create the FileTable.
- Every row in a FileTable represents a file or a directory.
- Every row contains the following items :

- A FILESTREAM column for stream data and a file_id (GUID) identifier. (The FILESTREAM column is NULL for a directory.)
- Both path_locator and parent_path_locator columns for representing and maintaining the file and directory hierarchy.
- 10 file attributes such as created date and modified date that are useful with file I/O APIs.
- A type column that supports full-text search and semantic search over files and documents.

➢ The FileTable feature builds on top of SQL Server FILESTREAM technology.

Filestream

➢ FILESTREAM enables SQL Server-based applications to store unstructured data, such as documents and images, on the file system.
➢ FILESTREAM is not automatically enabled when you install or upgrade SQL Server. You must enable FILESTREAM by using SQL Server Configuration Manager and SQL Server Management Studio.
➢ To use FILESTREAM, you must create or modify a database to contain a special type of filegroup. Then, create or modify a table so that it contains a varbinary(max) column with the FILESTREAM attribute.
➢ After you complete these tasks, you can use Transact-SQL and Win32 to manage the FILESTREAM data.

To know more about filestream please use the below link:

http://msdn.microsoft.com/en-us/library/gg471497(v=sql.110).aspx

File Tables extend the capabilities of the FILESTREAM feature of SQL Server. Therefore you have to enable FILESTREAM for file I/O access at the Windows level and on the instance of SQL Server before you can create and use File Tables.

To enable the prerequisites for creating and using FileTable, enable the following items:

1. Enable FILESTREAM at the Instance Level
2. Enable Non-Transactional Access at the Database Level
3. Specify a Directory for FileTable at the Database Level

Use the below link to know in detail about enabling the prerequisites

http://msdn.microsoft.com/en-us/library/gg509097(v=sql.110).aspx#EnablePrereq

DDL Operations on FileTable

Creating a FILETABLE

➢ Create a FileTable by Using Transact-SQL

We can create a FileTable by calling the CREATE TABLE (Transact-SQL) statement with the AS FileTable option. Since a FileTable has a fixed schema, there is no need to specify the list of columns. The two settings should be specified for the new FileTable:

FILETABLE_DIRECTORY (optional). Specifies the directory that serves as the root directory for all the files and directories stored in the FileTable. This name should be unique among all the FileTable directory names in the database. Comparison for uniqueness is case-insensitive, regardless of the current collation settings.

FILETABLE_COLLATE_FILENAME. Specifies the name of the collation to be applied to the Name column in the FileTable.

Example:

```
CREATE TABLE DocumentStore AS FileTable
  WITH (
      FileTable_Directory = 'DocumentTable',
      FileTable_Collate_Filename = database_default
     );
GO
```

➢ Create a FileTable by Using SQL Server Management Studio

In Object Explorer, expand the objects under the selected database, then right-click on the Tables folder, and then select New FileTable.

This option opens a new script window which contains a Transact-SQL script template that you can customize and run to create a FileTable. Use the Specify Values for Template Parameters option on the Query menu to customize the script easily.

32.1.1 Requirements and Restrictions for Creating a FileTable

- You cannot alter an existing table to convert it into a FileTable.
- The parent directory previously specified at the database level must have a non-null value.
- A FileTable requires a valid FILESTREAM filegroup, since a FileTable contains a FILESTREAM column.

- You cannot create a table constraint as part of a **CREATE TABLE…AS FILETABLE** statement. However you can add the constraint later by using an **ALTER TABLE** statement.
- You cannot create a FileTable in the **tempdb** database or in any of the other system databases.
- You cannot create a FileTable as a temporary table.

Altering a FileTable

Since a FileTable has a pre-defined and fixed schema, you cannot add or change its columns. However, you can add custom indexes, triggers, constraints, and other options to a FileTable.

32.1.2 Changing the Directory for a FileTable

Call the ALTER TABLE statement and provide a valid new value for the **FILETABLE_DIRECTORY** SET option.

```
ALTER TABLE filetable_name
    SET FILETABLE_DIRECTORY = N'directory_name';
GO
```

32.1.3 Requirements and Restrictions for Altering a FileTable

- You cannot change, drop, or disable the system-defined columns of a FileTable.
- You cannot add new user columns, computed columns, or persisted computed columns to a FileTable

Dropping a FileTable

You can drop a FileTable by using the ordinary syntax for the DROP TABLE (Transact-SQL) statement.

When you drop a FileTable, the following objects are also dropped:

- All the columns of the FileTable and all the objects associated with the table, such as indexes, constraints, and triggers, are also dropped.
- The FileTable directory and the sub-directories that it contained disappear from the FILESTREAM file and directory hierarchy of the database.

The DROP TABLE command fails if there are open file handles in the FileTable's file namespace.

Other Database Objects Are Created When You Create a FileTable

When you create a new FileTable, some system-defined indexes and constraints are also created. You cannot alter or drop these objects; they disappear only when the FileTable itself is dropped. To see the list of these objects, query the catalog view sys.filetable_system_defined_objects (Transact-SQL)

--View all objects for all File Tables, unsorted

```
SELECT * FROM sys.filetable_system_defined_objects;
GO
```

--View sorted list with friendly names

```
SELECT OBJECT_NAME(parent_object_id) AS 'FileTable',
OBJECT_NAME(object_id) AS 'System-defined Object'
    FROM sys.filetable_system_defined_objects
    ORDER BY FileTable, 'System-defined Object';
GO
```

When you create a new FileTable, the following system-defined indexes are also created:

Columns	**Index type**
[path_locator] ASC	Primary Key, non-clustered
[parent_path_locator] ASC, [name] ASC	Unique, non-clustered
[stream_id] ASC	Unique, non-clustered

Constraints that are created when you create a new FileTable

Constraints	Enforces

Default constraints on the following columns: • creation_time • is_archive • is_directory • is_hidden • is_offline • is_readonly • is_system • is_temporary • last_access_time • last_write_time • path_locator • stream_id	The system-defined default constraints enforce default values for the specified columns.
Check constraints	The system-defined check constraints enforce the following requirements: • Valid filenames. • Valid file attributes. • Parent object must be a directory. • Namespace hierarchy is locked during file manipulation.

Loading or Migrating Files into a FileTable

The method that you choose for loading or migrating files into a FileTable depends on where the files are currently stored.

Current location of files	Options for migration
Files are currently stored in the file system. SQL Server has no	Since a FileTable appears as a folder in the Windows file system, you can easily load files into a new FileTable by using any of the available methods for moving or copying files. These methods include Windows Explorer, command

knowledge of the files.	line options including xcopy and robocopy, and custom scripts or applications. You cannot convert an existing folder to a FileTable.
Files are currently stored in the file system. SQL Server contains a table of metadata that contains pointers to the files.	The first step is to move or copy the files by using one of the methods mentioned above. The second step is to update the existing table of metadata to point to the new location of the files.

32.1.4

- **Loading Files into a FileTable**

The methods that you can use to load files into a FileTable include the following:

- Drag and drop files from the source folders to the new FileTable folder in Windows Explorer.
- Use command line options such as MOVE, COPY, XCOPY, or ROBOCOPY from the command prompt or in a batch file or script.
- Write a custom application in C# or Visual Basic.NET that uses methods from the **System.IO** namespace to move or copy the files.

- **Bulk Load Files into a FileTable**

You can use various methods to bulk load files into a FileTable:

- Bcp

Call with the **CHECK_CONSTRAINTS** clause.
Disable the FileTable namespace and call without the **CHECK_CONSTRAINTS** clause. Then re-enable the FileTable namespace.

- BULK INSERT

Call with the **CHECK_CONSTRAINTS** clause.

Disable the FileTable namespace and call without the **CHECK_CONSTRAINTS** clause. Then re-enable the FileTable namespace.

- INSERT INTO … SELECT * FROM OPENROWSET (BULK …)

Call with the **IGNORE_CONSTRAINTS** clause.
Disable the FileTable namespace and call without the **IGNORE_CONSTRAINTS** clause. Then re-enable the FileTable namespace.

For more details please go to the below link
http://msdn.microsoft.com/en-us/library/gg492083(v=sql.110).aspx

DML operations on FileTable

INSERT Operations on File Tables

The following considerations apply to INSERT Operations on File Tables:

- All the file attribute columns have NOT NULL constraints. If values are not explicitly set, then appropriate default values are supplied.
- System-defined constraints are enforced if the INSERT statement sets the **name**, **path_locator**, **parent_path_locator**, or file attributes.
- The application can obtain the **path_locator** for a file or directory by providing the file system path to the GetPathLocator (Transact-SQL) function.

UPDATE Operations on File Tables

The following considerations apply to **UPDATE** operations on File Tables:

- Updates to any user-defined data are allowed.
- System-defined constraints are enforced if the INSERT statement sets the **name**, **path_locator**, **parent_path_locator**, or file attributes.
- Updates can be made to the FILESTREAM data in the **file_stream** column without affecting any of the other columns, including the timestamps.

DELETE Operations on File Tables

The following considerations apply to **DELETE** operations on File Tables:

- Deleting a row also removes the corresponding file or directory from the file system.
- Deleting a row fails if the row corresponds to a directory that contains other files or directories.

Constraints That Are Enforced for DML Operations on File Tables

System-defined constraints ensure that DML actions do not compromise the integrity of the file namespace hierarchy. The constraints that are enforced include the following:

- When you set or change the name of the file or directory:
  - Windows file and directory naming conventions are enforced.
  - The uniqueness of the name in the parent directory is enforced.
- When you set or change the location of a file or directory by setting or changing the path_locator or parent_path_locator:
  - Uniqueness is enforced.
  - The consistency of the hierarchical tree of directories and files is enforced, including the consistency of path_locator and parent_path_locator values.
- The value of is_directory cannot be set to true when the file_stream column is not null. Data in the file_stream column indicates that the row represents a file and not a directory.
- File attribute columns cannot be null. NOT NULL constraints are enforced with default values.
- The value of last_access_time cannot be earlier than last_write_time and creation_time.

Get a List of File Tables and Related Objects

To get a list of File Tables, query one of the following catalog views:

- sys.filetables (Transact-SQL)
- sys.tables (Transact-SQL) (Check the value of the **is_filetable** column.)

```
SELECT * FROM sys.filetables;
GO

SELECT * FROM sys.tables WHERE is_filetable = 1;
GO

SELECT object_id, OBJECT_NAME(object_id) AS 'Object Name'
FROM sys.filetable_system_defined_objects
WHERE object_id = filetable_object_id;
GO
```

32.1.5 Identify the Locks Held by File Tables

Join the **request_owner_id** field in the dynamic management view sys.dm_tran_locks (Transact-SQL) with the **fcb_id** field in sys.dm_filestream_non_transacted_handles (Transact-SQL). In some cases, the lock does not correspond to a single open file handle.

```
SELECT opened_file_name
FROM sys.dm_filestream_non_transacted_handles
WHERE fcb_id IN
(SELECT request_owner_id FROM sys.dm_tran_locks);
GO
```

Monitoring FileTable Activity by Using SQL Server Profiler

SQL Server Profiler can capture the Windows File Open and File Close operations in trace output for files that are stored in a FileTable.

SQL Server Features and File Tables

- **Partitioning and File Tables**

Partitioning is not supported on File Tables. With the support for multiple FILESTREAM file groups, pure scale-up issues can be handled without having to resort to partitioning in most scenarios (unlike SQL 2008 FILESTREAMs).

- **Replication and File Tables**

Replication and related features (including transactional replication, merge replication, change data capture, and change tracking) are not supported with File Tables.

- **Transaction Semantics and File Tables**

Windows applications do not understand database transactions, so Windows write operations do not provide the ACID properties of a database transaction. Therefore transactional rollbacks and recovery are not possible with Windows update operations.

- Transact-SQL applications

For TSQL applications working on the FILESTREAM (file_stream) column in a FileTable, the isolation semantics are the same as with FILESTREAM data type in a regular user table.

- **Query Notifications and File Tables**

The query cannot contain reference to the FILESTREAM column in the FileTable, in the WHERE clause or any other part of the query.

- **SELECT INTO and File Tables**

SELECT INTO statements from a FileTable will not propagate the FileTable semantics on the created destination table (just like FILESTREAM columns in a regular table). All the destination table columns will behave just like normal columns. They will not have any FileTable semantics associated with them.

- **Triggers and File Tables**

➢ DDL (Data Definition Language) Triggers
There are no special considerations for DDL triggers with File Tables.
➢ DML (Data Manipulation Language) Triggers

These restrictions will be enforced during the DML operation to create triggers.

- File Tables will NOT support INSTEAD OF triggers for DML operations. This is an existing restriction on all tables that contain FILESTREAM columns.
- File Tables will support AFTER triggers for DML operations.
- Triggers defined on a FileTable cannot update any File Tables (including the parent FileTable). This restriction exists mainly to prevent a trigger from getting into locking conflicts with the locks held by the file system access in the same transaction.

- **Views and File Tables**

A view can be created on a FileTable as on any other table. However the following considerations apply to a view created on a FileTable:

✓ View will not have any FileTable semantics. i.e. the columns in the view (including File Attribute columns) behave just like normal view columns without any special semantics and same is true for rows representing Files/directories.
✓ View may be updateable based on the "updateable view" semantics, but the underlying table constraints can reject the updates just like in the table.
✓ File path for a file can be visualized in the view by adding it as an explicit column in the view. For example:

```
CREATE VIEW MP3FILES AS SELECT column1, column2, …, GetFileNamespacePath() AS
PATH, column3,… FROM Documents
```

- Indexed Views

  Currently Indexed views cannot include FILESTREAM columns or computed/persisted computed columns that depend on the FILESTREAM columns. This behavior remains unchanged with views defined on the FileTable also.

- **Snapshot Isolation and File Tables**

Read Committed Snapshot Isolation (RCSI) and Snapshot Isolation (SI) rely on the ability to have a snapshot of the data available for readers even when update operations are happening on the data. However File Tables allow non-transactional write access to Filestream data. As a result, the following restrictions apply to the use of these features in databases that contain File Tables:

- ✓ A database that contains File Tables can be altered to enable RCSI/SI.
- ✓ When the non_transactional access is set to FULL for the database, then a transaction is running under RCSI or SI has the following behavior:
  - o Any TSQL reads to FileTable file_stream column would fail. INSERT and UPDATE to the column would still succeed, as long as it doesn't read from the file_stream column.
  - o If the TSQL query specifies READCOMMITTEDLOCK table hints, reads will succeed, and will take locks on the rows, rather than use row versioning.
  - o Transacted Win32 FileStream open would also fail.
  - o Non-transacted FileTable Win32 access will succeed. All internal queries done by FileTable won't be affected.
  - o Full text indexing will always succeed, no matter what the database options are (READ_COMMITTED_SNAPSHOT, ALLOW_SNAPSHOT_ISOLATION).

- **Contained Databases and File Tables**

Databases that contain File Tables are not fully contained because of their dependency on the share that is specified at the level of the SQL Server instance for FILESTREAM data.

- **DBCC and File Tables**

You can use DBCC CHECKCONSTRAINTS to validate the constraints on a FileTable including system-defined constraints.

FileTable DDL, Functions, Stored Procedures, and Views

Transact-SQL statements and the SQL Server database objects that have been added or changed to support the FileTable feature are

Transact-SQL Data Definition Language (DDL) Statements

Object	Status	More Information
ALTER DATABASE (Transact-SQL) ALTER DATABASE SET Options (Transact-SQL)	Changed	Enable the Prerequisites for FileTable Manage File Tables
ALTER TABLE (Transact-SQL)	Changed	Create, Alter, and Drop File Tables Manage File Tables
CREATE DATABASE (Transact-SQL)	Changed	Enable the Prerequisites for FileTable
CREATE TABLE (Transact-SQL)	Changed	Create, Alter, and Drop File Tables
RESTORE (Transact-SQL) RESTORE Arguments (Transact-SQL)	Changed	

Functions

Object	Status	More Information
FileTableRootPath (Transact-SQL)	**Added**	Work with Directories and Paths in File Tables
GetFileNamespacePath (Transact-SQL)	**Added**	Work with Directories and Paths in File Tables
GetPathLocator (Transact-SQL)	**Added**	Work with Directories and Paths in File Tables

Stored Procedures

Object	Status	More Information
sp_kill_filestream_non_transacted_handles (Transact-SQL)	**Added**	Manage File Tables

Catalog Views

Object	Status	More Information
sys.database_filestream_options (Transact-SQL)	**Added**	Enable the Prerequisites for FileTable
sys.filetable_system_defined_objects (Transact-SQL)	**Added**	Create, Alter, and Drop File Tables Manage File Tables
sys.filetables (Transact-SQL)	**Added**	Manage File Tables
sys.tables (Transact-SQL)	Changed	Manage File Tables

Dynamic Management Views

Object	Status	More Information
sys.dm_filestream_non_transacted_handles (Transact-SQL)	**Added**	Manage File Tables

33 References

http://msdn.microsoft.com/en-us/library/ms189461(v=sql.110).aspx
http://msdn.microsoft.com/en-us/library/ms189798(v=sql.110).aspx
http://msdn.microsoft.com/en-us/library/ms173454(v=sql.110).aspx
http://msdn.microsoft.com/en-us/library/ff878370(v=sql.110).aspx
http://msdn.microsoft.com/en-us/library/hh213234(v=sql.110).aspx
http://msdn.microsoft.com/en-us/library/cc645581(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/cc645579(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/cc645577(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/cc645580(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/cc645578(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/ee677615(v=sql.110).aspx
http://sqlblog.com/blogs/aaron_bertrand/archive/2010/11/11/sql-server-11-denali-using-sequence.aspx
http://blogs.msdn.com/b/ssma/archive/2011/07/12/converting-oracle-sequence-using-ssma-for-oracle-v5-1.aspx
http://www.codeproject.com/KB/database/Denali_Tsql_Part_2.aspx?display=Print#3
http://www.nielsberglund.com/sql/new-t-sql-features-in-sql-11-denali-error-handling/
http://www.nielsberglund.com/sql/more-t-sql-error-functionality-in-denali-sql-11/
http://msdn.microsoft.com/en-us/library/cc645577(v=SQL.110).aspx
http://msdn.microsoft.com/en-us/library/ms188385(v=SQL.110).aspx