SQL Server 2014/2016 Enhancements for Developers

Wylie Blanchard
Lead IT Consultant; SQL Server DBA

GTP GREAT TECH PROS

Please Support Our Sponsors

SQL Saturday is made possible with the generous support of these sponsors.
You can support them by opting-in and visiting them in the sponsor area.

Don’t Forget

- Silence your cell phones
- Online Evaluations
  - www.sqlsaturday.com/572/sessions/sessionevaluation.aspx
  - www.sqlsaturday.com/572/eventeval.aspx
- Submit for raffles by 3:30PM

About Great Tech Pros

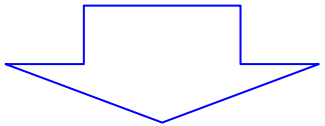
- Great Tech Pros was founded in 2012
- Specialties include:
  - IT Consulting
  - Database Administration, Management
  - Data Analysis
  - Website Design and Development
  - Professional Training and Presentations
- Visit us at www.GreatTechPros.com

GTP GREAT TECH PROS

Speaker

Wylie Blanchard

- SQL Server Database Consultant
- MCSE: SQL Server Data Platform
- Website: WylieBlanchard.com
- LinkedIn: in/WylieBlanchard
- Twitter: @WylieBlanchard1



- Pizza Connoisseur (self proclaimed)

Presentation Summary

Learn what's new in SQL 2014/2016. Which features and enhancements are really important to the work life of a SQL Server Developer.

In this presentation we'll explore SQL Server 2014/2016 new possibilities, showing you how to use new T-SQL functions, features and enhancements that are only available in SQL Server 2014/2016.

What's New - SQL Server 2014/2016

- In-Memory OLTP
- Drop If Exists
- Select … Into
- Dynamic Data Masking
- Query Store

Memory-Optimized Tables

In-Memory OLTP

- Memory Optimized Tables
  - Tables using the new data structures
- Allow highly used tables to live in memory
  - Remain in memory forever without losing records
- Designed to reduce blocking and locks
- High Performance response than disk tables due to data living in memory

In-memory OLTP - Demo

Steps:

1. Create Database Which Creates A File Group Containing Memory_Optimized_Data
2. Create two different tables 1) Regular table and 2) Memory Optimized table
3. Create two stored procedures 1) Regular SP and 2) Natively Compiled SP
4. Compare the performance of two SPs

In-memory OLTP - Demo (cont)

```
/** Create Database Which Creates A File Group Containing Memory_Optimized_Data **/
 Use master
/** create database **/
CREATE DATABASE InMemOLTP
ON PRIMARY(NAME = InMemOLTPData,
FILENAME = 'c:\data\InMemOLTPData.mdf', size=200MB),
/** memory optimized data **/
FILEGROUP [InMemOLTP_FG] CONTAINS MEMORY_OPTIMIZED_DATA(
NAME = [InMemOLTP_InMemOLTP_dir],
FILENAME = 'c:\data\InMemOLTP_InMemOLTP_dir')
LOG ON (name = [InMemOLTP_demo_log], Filename='c:\data\InMemOLTP.ldf', size=100MB)
GO
```

In-memory OLTP - Demo (cont)

```
/** Create A Regular Table & A Table With Setting Memory_Optimized Set To Enabled **/
USE InMemOLTP
GO
/** create a regular table **/
CREATE TABLE RegularTable (ID INT NOT NULL PRIMARY KEY,
Name VARCHAR(100) NOT NULL)
GO
/** create a memory optimized table **/
CREATE TABLE MemoryTable (ID INT NOT NULL,
Name VARCHAR(100) NOT NULL
CONSTRAINT ID_Clust_MemoryTable PRIMARY KEY NONCLUSTERED HASH (ID) WITH
(BUCKET_COUNT=1000000))
WITH (MEMORY_OPTIMIZED=ON)
GO
```

In-memory OLTP - Demo (cont)

```
/** Create A Regular Stored Procedure - simple table to insert 100,000 rows **/
CREATE PROCEDURE Reglar_Insert_test
AS
 BEGIN
 SET NOCOUNT ON
 DECLARE @counter AS INT = 1
DECLARE @start DATETIME
 SELECT @start = GETDATE()
WHILE (@counter <= 100000)
BEGIN
 INSERT INTO RegularTable VALUES(@counter, 'WylieBlanchard')
SET @counter = @counter + 1
END
 SELECT DATEDIFF(SECOND, @start, GETDATE() ) [Regular_Insert in sec]
END
GO
```

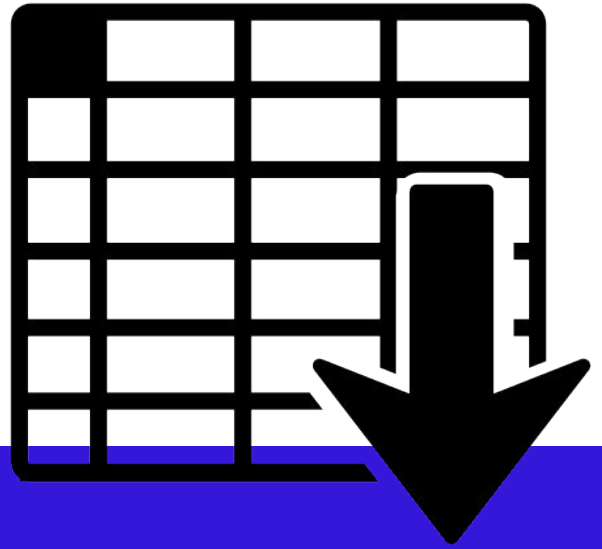
In-memory OLTP - Demo (cont)

```
/** Create A Natively Compiled Stored Procedure InMemOLTP table to insert 100,000 **/
CREATE PROCEDURE ImMemory_Insert_test
WITH NATIVE_COMPILATION, SCHEMABINDING,EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='english')
DECLARE @counter AS INT = 1
DECLARE @start DATETIME
SELECT @start = GETDATE()
WHILE (@counter <= 100000)
BEGIN
INSERT INTO dbo.MemoryTable VALUES(@counter, 'WylieBlanchard')
SET @counter = @counter + 1
END
SELECT DATEDIFF(SECOND, @start, GETDATE() ) [InMemOLTP_Insert in sec] END
GO
```

In-memory OLTP - Demo (cont)

```
/** Compare the Performance of both Stored Procedures **/

/** Insert data into [RegularTable] **/
EXEC Reglar_Insert_test
GO
/** Insert data into [MemoryTable] **/
EXEC ImMemory_Insert_test
GO
```

DROP IF EXISTS



DROP Statements

- Use DROP statements to remove existing entities.
  - For example, use DROP TABLE to remove a table from a database.
  - Syntax has been enhanced for verifying whether the entity exists when dropping.

Example: DROP Statement Syntax

```
-- Syntax for SQL Server and Azure SQL Database

DROP TABLE [ IF EXISTS ] [ database_name . [
schema_name ] . | schema_name . ]
table_name [ ,...n ]
```

DROP IF EXISTS - Demo

```
/** Old Drop Procedure Script **/
IF EXISTS (SELECT * FROM sys.procedures WHERE name = 'Reglar_Insert_test')
DROP PROCEDURE Reglar_Insert_test

/** New 2016 Drop Procedure Script **/
DROP PROCEDURE IF EXISTS [dbo].[ImMemory_Insert_test]

/** Old Drop Table Script **/
IF OBJECT_ID('[dbo].[RegularTable]', 'U') IS NOT NULL
DROP TABLE [dbo].[RegularTable];

/** New 2016 Drop Table Script **/
DROP TABLE IF EXISTS  [dbo].[MemoryTable]
```

DROP IF EXISTS - Objects

- AGGREGATE
- PROCEDURE
- TABLE
- ASSEMBLY
- ROLE
- TRIGGER
- VIEW
- RULE
- DATABASE
- SCHEMA USERDEFAULT
- SECURITY POLICY
- VIEW
- FUNCTION
- SEQUENCE
- INDEX
- TYPE
- SYNONYM

SELECT … INTO

SELECT … INTO

The SELECT … INTO statement is improved and can now operate in parallel. The database compatibility level must be at least 110.

- Improves performance by utilizing multiple processor cores.

SELECT … INTO - Demo

```
/** Change Compatibility Level to SQL Server 2008 **/
USE [master]
GO
ALTER DATABASE [TestDB] SET COMPATIBILITY_LEVEL = 100
GO

/** Insert rows into a new table **/
SELECT  [ID], [Name], SUM(ID) as Total  INTO [TestDB].[dbo].[NewTable01] from
[TestDB].[dbo].[TestTable1]
group by [ID], [Name]
GO

/** Review Execution Plan **/
```

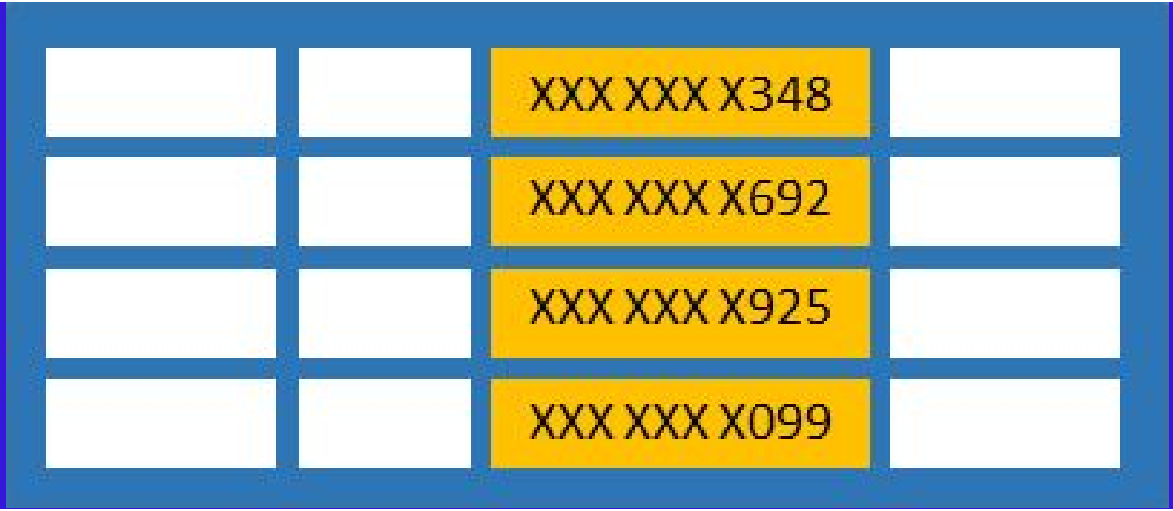
SELECT … INTO - Demo (Cont)

```
/** Change Compatibility Level to SQL Server 2014 or 2016 **/
USE [master]
GO
ALTER DATABASE [TestDB] SET COMPATIBILITY_LEVEL = 120
GO

/** Inset rows into a new table **/
SELECT  [ID], [Name], SUM(ID) as Total  INTO [TestDB].[dbo].[NewTable02] from
[TestDB].[dbo].[TestTable1]
group by [ID], [Name]
GO

/** Review Execution Plan **/
```

SELECT … INTO - Demo (Cont)

```
/** Change Compatibility Level to SQL Server 2016 **/
USE [master]
GO
ALTER DATABASE [TestDB] SET COMPATIBILITY_LEVEL = 130

/** Clean Up - Drop tables **/
drop table if exists [TestDB].[dbo].[NewTable01]
drop table if exists [TestDB].[dbo].[NewTable02]
```

Dynamic Data Masking (DDM)



Dynamic Data Masking

- Limits sensitive data exposure by masking it to non-privileged users
- Helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal impact on the application layer.
- It’s a policy-based security feature that hides the sensitive data in the result set of a query over designated database fields, while the data in the database is not changed

Dynamic Data Masking - Demo

```
SET NOCOUNT ON
GO
/** drop database MaskingDatabase - if already exists **/
USE [master]
GO
DROP DATABASE IF EXISTS [MaskingDatabase]

/** create new database called MaskingDatabase **/
CREATE DATABASE MaskingDatabase
GO
USE MaskingDatabase
GO
```

Dynamic Data Masking - Demo (Cont)

```
/** Create table with different data type columns **/
CREATE TABLE AccountInfo (
ID INT IDENTITY(1, 1) PRIMARY KEY
,fName NVARCHAR(30) NOT NULL ,lName NVARCHAR(30) NOT NULL
,CreditCard VARCHAR(20) NULL ,CVC INT NULL
,AccountEmail NVARCHAR(60) NULL ,PersonalEmail NVARCHAR(60) NULL
,CurrentDate DATETIME NULL
)
/** insert a row **/
INSERT INTO [dbo].[AccountInfo]
([fName],[lName] ,[CreditCard],[CVC],[AccountEmail],[PersonalEmail], [CurrentDate])
VALUES('Blanchard','Wylie','1234-5678-1234-5678',1234,'wblanchard@GreatTechPros.com',
'wylieblanchard@gmail.com', '10-September-2016')
GO
```

Dynamic Data Masking - Demo (Cont)

```
 /** apply masking **/
ALTER TABLE AccountInfo
ALTER COLUMN CreditCard ADD MASKED WITH (FUNCTION = 'partial(2,"XX-XXXX-XXXX-XX",2)')
ALTER TABLE AccountInfo
ALTER COLUMN CVC ADD MASKED WITH (FUNCTION = 'default()')-- default on int
ALTER TABLE AccountInfo
ALTER COLUMN CurrentDate ADD MASKED WITH (FUNCTION = 'default()')-- default on date
ALTER TABLE AccountInfo
ALTER COLUMN fname ADD MASKED WITH (FUNCTION = 'default()')-- default on varchar
ALTER TABLE AccountInfo
ALTER COLUMN AccountEmail ADD MASKED WITH (FUNCTION = 'email()')
GO
```

Dynamic Data Masking - Demo (Cont)

```
/** create a new user and grant select permissions **/
USE MaskingDatabase
GO
CREATE USER WhoAmI WITHOUT LOGIN;
GRANT SELECT ON AccountInfo TO WhoAmI;

/** Example selecting the data as user**/
USE MaskingDatabase
GO
SELECT * FROM AccountInfo; -- this would show clear data
GO
EXECUTE AS USER = 'WhoAmI';
SELECT * FROM AccountInfo -- this should show masked data
REVERT;
GO
```

DDM Practices & Uses

- Creating a mask on a column does not prevent updates to that column.
- Using SELECT INTO or INSERT INTO to copy data from a masked column into another table results in masked data in the target table.
- Dynamic Data Masking is applied when running SQL Server Import and Export.

DDM - Limitations

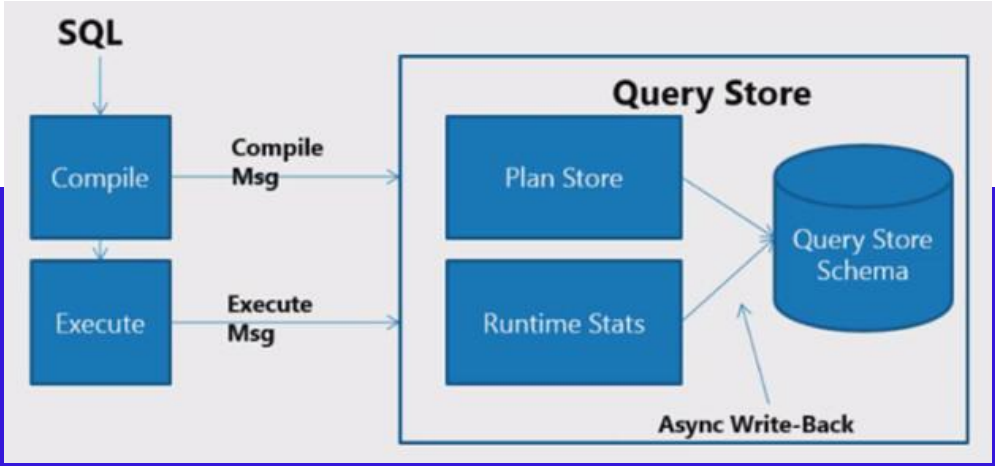
Masking rule can’t be used for the following column types:

- Encrypted columns (Always Encrypted)
- FILESTREAM
- COLUMN_SET or a sparse column that is part of a column set.

DDM - Limitations (Cont)

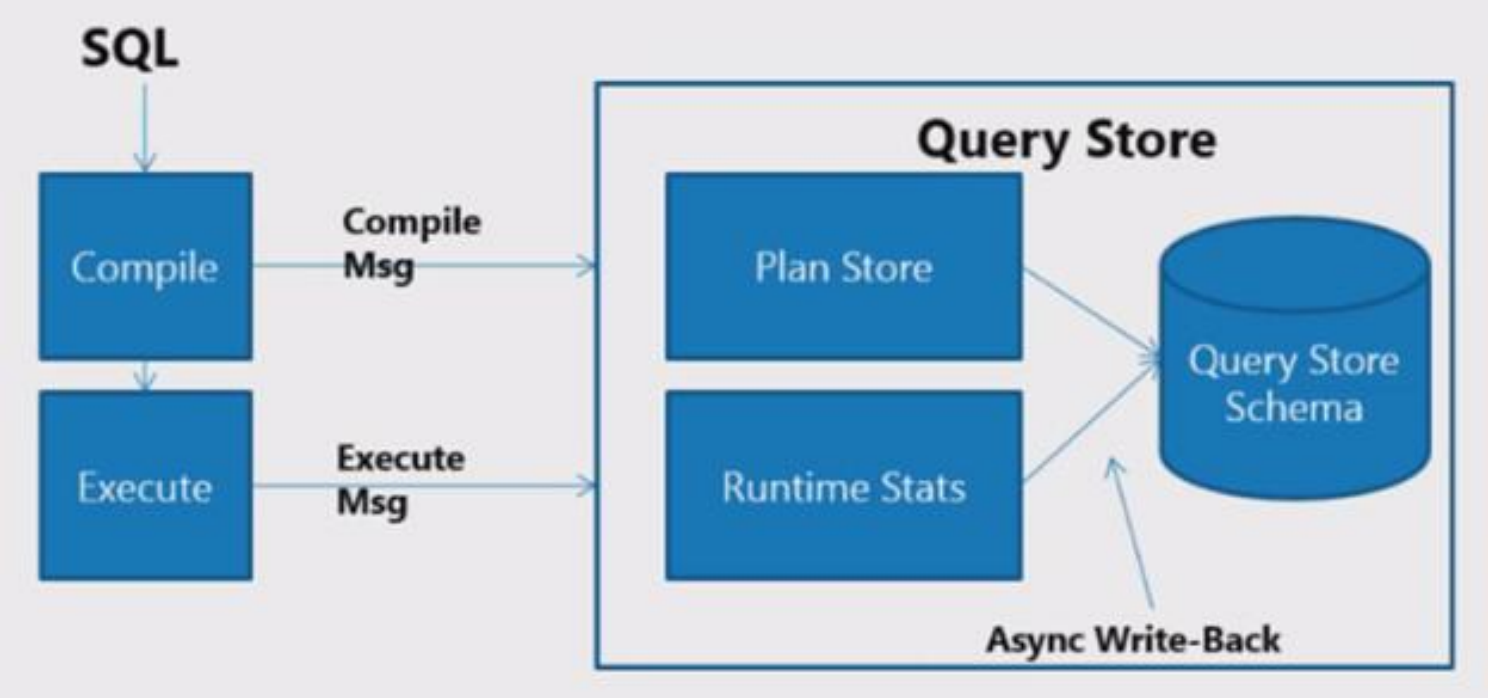
- A mask cannot be configured on a computed column, but if the computed column depends on a column with a MASK, then the computed column will return masked data.
- A column with data masking cannot be a key for a FULLTEXT index

Query Store


SQL
Compile
Execute
Compile Msg
Execute Msg
Query Store
Plan Store
Runtime Stats
Query Store Schema
Async Write-Back

Using the Query Store

- Helps simplify performance troubleshooting
- Capture queries, query plans, run time stats, etc.
- New system views created for Query Store
- View a history of Query workload


SQL
Compile
Execute
Compile
Msg
Execute
Msg
Query Store
Plan Store
Runtime Stats
Query Store
Schema
Async Write-Back

Enabling the Query Store

Using the Query Store Page in Management Studio

1. In Object Explorer, right-click a database, and then click **Properties**.
2. In the **Database Properties** dialog box, select the **Query Store** page.
3. In the **Operation Mode (Requested)** box, select **On**.

Enabling the Query Store (Cont)

Use the ALTER DATABASE statement to enable the query store. For example

```
/**Enabling the Query Store**/

Use [TestDB]
ALTER DATABASE TestDB SET QUERY_STORE = ON;
```

Query Store Demo

```
/**Create Demo Query Store Database**/
use master
CREATE DATABASE [qstore_demo]  ON  PRIMARY
( NAME = N'qs_demo', FILENAME = N'C:\DATA\qs_demo.mdf' , SIZE = 102400KB ,
              MAXSIZE = 1024000KB , FILEGROWTH = 20480KB )
 LOG ON
( NAME = N'qs_demo_log', FILENAME = N'c:\DATA\qs_demo_log.ldf' , SIZE = 20480KB ,
              MAXSIZE = 1024000KB , FILEGROWTH = 20480KB )
GO ALTER DATABASE [qstore_demo] SET AUTO_UPDATE_STATISTICS OFF
GO ALTER DATABASE [qstore_demo] SET AUTO_CREATE_STATISTICS OFF
GO ALTER DATABASE [qstore_demo] SET RECOVERY SIMPLE
GO ALTER DATABASE [qstore_demo] SET QUERY_STORE = OFF
GO
```

Query Store Demo (Cont)

```
/**Create table sp and populate table**/
USE qstore_demo
GO
-- create a table
CREATE TABLE dbo.db_store (c1 CHAR(3) NOT NULL, c2 CHAR(3) NOT NULL, c3 SMALLINT
NULL)
GO
-- create a stored procedure
CREATE PROC dbo.proc_1 @par1 SMALLINT
AS
SET NOCOUNT ON
SELECT c1, c2 FROM dbo.db_store
WHERE c3 = @par1
GO
```

Query Store Demo (Cont)

```
 -- populate the table (this may take a couple of minutes)
SET NOCOUNT ON
INSERT INTO [dbo].db_store (c1,c2,c3) SELECT '18','2f',2
go 20000
INSERT INTO [dbo].db_store (c1,c2) SELECT '171','1ff'
go 4000
INSERT INTO [dbo].db_store (c1,c2,c3) SELECT '172','1ff',0
go 10
INSERT INTO [dbo].db_store (c1,c2,c3)   SELECT '172','1ff',4
go 15000
-- enable Query Store on the database
ALTER DATABASE [qstore_demo] SET QUERY_STORE = ON
GO
```

Query Store Demo (Cont)

```
/** Test 1 - No Indexes on the Table **/
EXEC dbo.proc_1 0
GO 20

--Review Query Store "Top Resource Consuming Queries"

/**Test 2 - Testing with a Non Clustered Index**/
CREATE NONCLUSTERED INDEX NCI_1
ON dbo.db_store (c3)
GO
EXEC dbo.proc_1 0
GO 20

--Review Query Store "Top Resource Consuming Queries"
```

Query Store Demo (Cont)

```
--Clean up

ALTER DATABASE [qstore_demo] SET QUERY_STORE = ON;

USE master
drop database [qstore_demo]
```

Other Features

Not mentioned in this presentation but are worth researching.

Live Query Statistics
Native JSON Support
Polybase
Advance Analytics
BI on Mobile Devices
Data Stretch to MS Azure

Resource Links

- In-Memory OLTP (In-Memory Optimization)
- SQL SERVER – Beginning In-Memory OLTP with Sample Example
- DROP IF EXISTS – new thing in SQL Server 2016
- SQL Server 2014 New Features: Parallel SELECT INTO
- Exploring SQL Server 2014 SELECT INTO Parallelism
- Dynamic Data Masking
- SQL Server 2016 Query Store Example

Questions

Connect With Us

- Twitter: @GreatTechPros
- Linkedin: /company/Great-Tech-Pros
- Google+: +GreatTechPros
- Facebook: /GreatTechPros
- Website: GreatTechPros.com

Thank You

This FREE SQL Saturday is brought to you courtesy of these sponsors, speakers and volunteers who staff this event