SQL Server 2014/2016 Enhancements for Developers

Wylie Blanchard
Lead IT Consultant; SQL Server DBA

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

Speaker

Wylie Blanchard

- SQL Server Database Consultant
- MCSE: SQL Server Data Platform
- Website: WylieBlanchard.com
- LinkedIn: in/WylieBlanchard
- Twitter: @WylieBlancahrd1

- Pizza Connoisseur (self proclaimed)

Presentation Summary

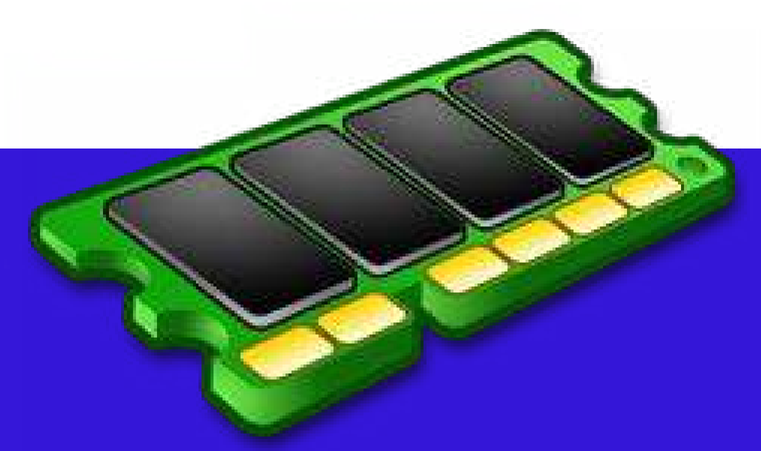
Learn what's new in SQL 2014/2016. Which features and enhancements are really important to the work life of a SQL Server Developer.

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 (2014)
- Drop If Exists (2016)
- Select … Into (2014)
- Dynamic Data Masking (2016)

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.

SELECT … INTO - Demo

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

/** Inset 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
```

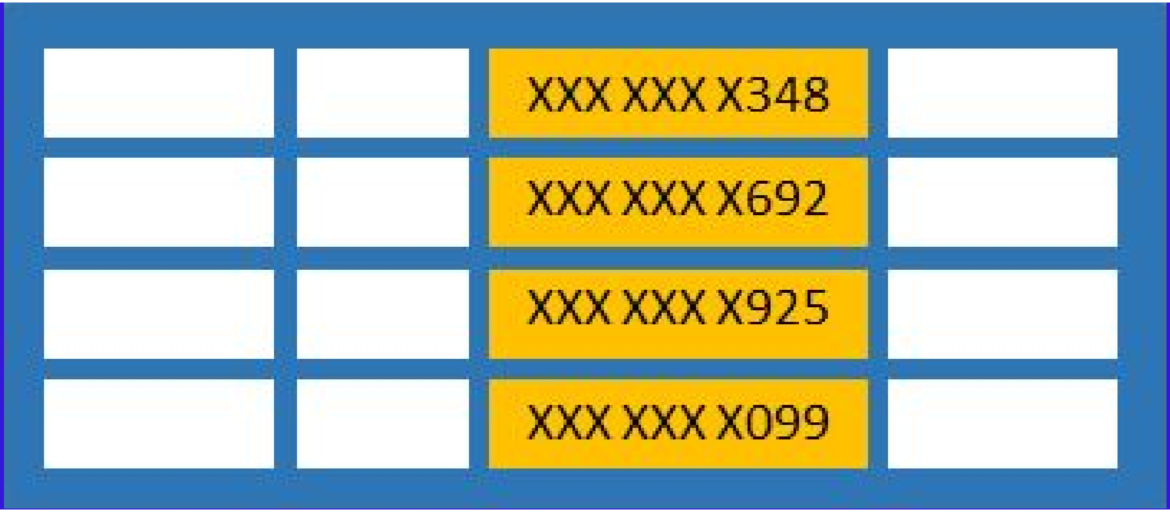
SELECT ... INTO - Demo (Cont)

```
/** 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
```

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)



		XXX XXX X348	
		XXX XXX X692	
		XXX XXX X925	
		XXX XXX X099	

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

```sql
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)

```sql
/** 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

Other Features

Not mentioned in this presentation but are worth researching.

Data Quality Services
Data Master Services
SQL Server Integration Services

- SSIS - Undo and Redo Features
- SSIS - Project Deployments

SQL Server Reporting Services

- SSRS Power View
- SSRS Data Alerts
- PowerPivot (not new)
- Google Chrome enhancements (2014)

Azure integration enhancements

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
- Exploring SQL Server 2014 SELECT INTO Parallelism
- Dynamic Data Masking

Thank You

Connect With Us

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

Bonus Content

Contained Databases

Contained Databases

- A database that is independent from the SQL Server instance
    - Has the ability to maintain database description metadata in the database
    - It can also perform user authentication at database level in addition to (or instead of) the server level

Contained Databases (cont)

Benefits

1. User authentication can be performed by the database
   - reduces the databases dependency on the logins of the instance of SQL Server
2. Easily move a database from one instance of SQL Server to another
   - Metadata maintained in actual database instead of the master database

Contained Databases (cont)

Benefits (cont)

3. Give db owner more control over database, without giving the db owner sysadmin permission
    - Errors related to missing users and orphan users are no longer an issue with contained databases

Contained Databases (cont)

Disadvantages

1. DB Owner can create contained db users without the permission of a DBA
   - can lead to security issues & data theft threat
2. Can’t use replication, change data capture, change tracking, numbered procedures, schema-bound objects that depend on built-in functions with collation changes

Contained Databases (cont)

Disadvantages (cont)

3. A user confined to the contained database may be able to access other databases on the Database Engine
    - if the other databases have enabled the guest account

Contained DB - Demo

Steps:

1. Enable database at the server/instance level
2. Enable containment at the database level
3. Create a contained user
4. Test connectivity

Contained DB - Demo (cont)

```
/** enable database containment on server-instance **/
sp_configure 'show advanced options', 1
GO
RECONFIGURE WITH OVERRIDE
GO
sp_configure 'contained database authentication', 1
GO
RECONFIGURE WITH OVERRIDE
GO
sp_configure 'show advanced options', 0
GO
RECONFIGURE WITH OVERRIDE
GO
```

Contained DB - Demo (cont)

```
/** enable contained database on database **/
USE [master]
 GO
ALTER DATABASE [MaskingDatabase] SET CONTAINMENT = PARTIAL WITH NO_WAIT
 GO

/** create a contained user **/
USE [MaskingDatabase]
 GO
CREATE USER [ContainedUser] WITH PASSWORD=N'Password', DEFAULT_SCHEMA=[dbo]
 GO

/** test connectivity **/
```

Contained DB - Demo (cont)

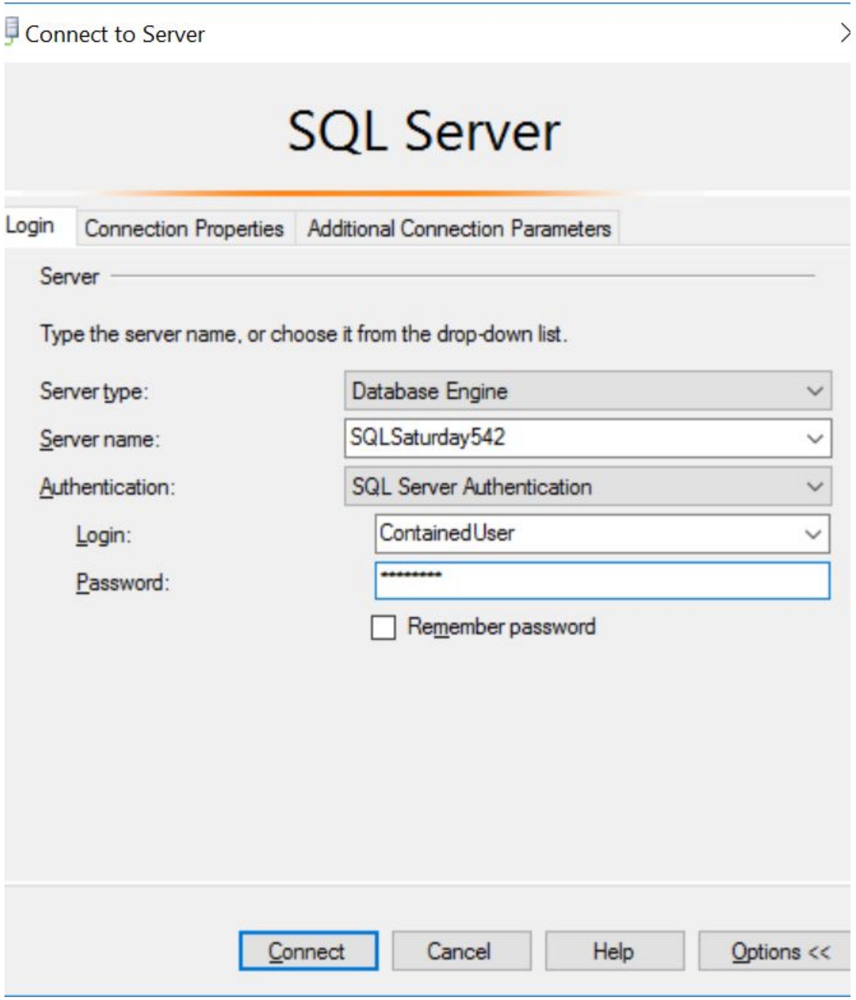
Test Connectivity

1. Close and reopen SSMS
2. Click “Options” once the login screen appears
3. Select ‘Database Engine’ for “Server type”
4. Specify the instance that hosts the database for “Server Name”

Contained DB - Demo (cont)

Test Connectivity (cont)

5. Enter the user login credentials that were created (Do not click Connect)
6. Navigate to the “Connection Properties” tab
7. Specify the name of the contained database in the “Connect to Database” box
8. Click “Connect”


Connect to Server

SQL Server

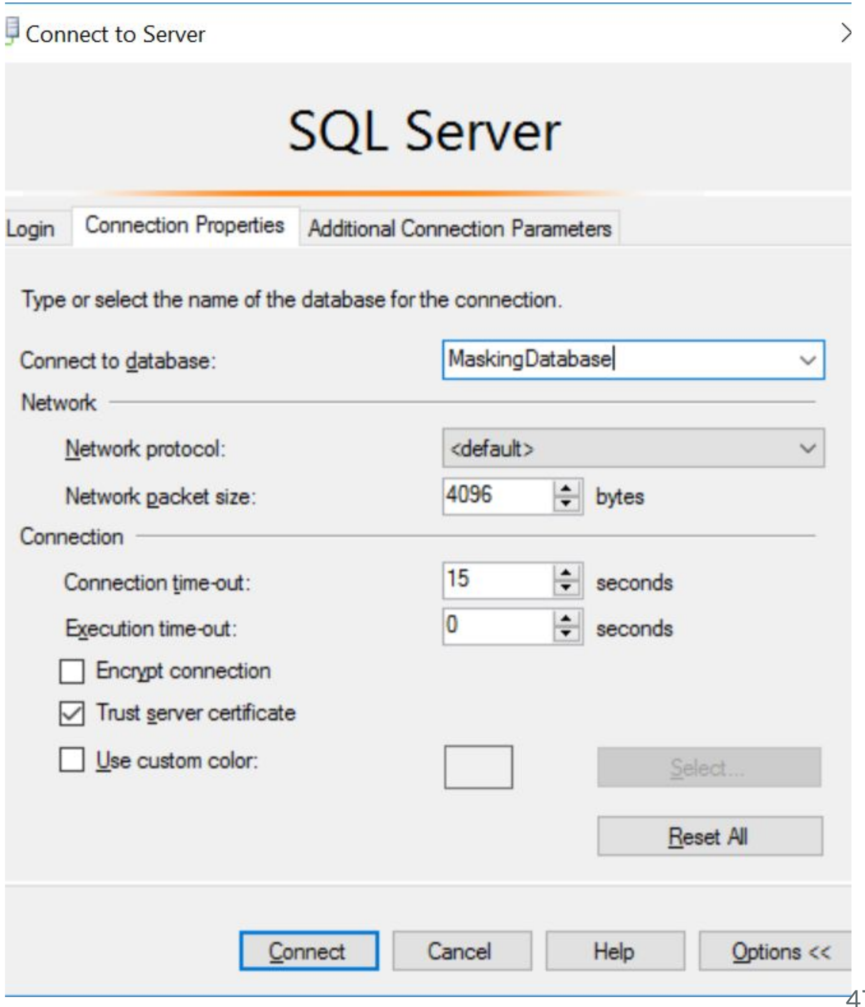
Login | Connection Properties | Additional Connection Parameters

Server

Type the server name, or choose it from the drop-down list.

Server type:	Database Engine
Server name:	SQLSaturday542
Authentication:	SQL Server Authentication
Login:	ContainedUser
Password:	*********

☐ Remember password


Connect to Server

SQL Server

Login | Connection Properties | Additional Connection Parameters

Type or select the name of the database for the connection.

Connect to database: MaskingDatabase

Network

Network protocol:	<default>
Network packet size:	4096 bytes

Connection

Connection time-out:	15 seconds
Execution time-out:	0 seconds

☐ Encrypt connection

☑ Trust server certificate

☐ Use custom color:

Contained DB - Resource Links

Contained Databases -

https://msdn.microsoft.com/en-us/library/ff929071.aspx