IDERA® | SQL Server eBook

SQL SERVER IN SIMPLE WORDS

BY PINAL DAVE

Learn tips and techniques for using SQL Server Management Studio (SSMS)

TABLE OF CONTENTS

SECTION ONE : ZERO TO HERO WITH SQL SERVER MANAGEMENT STUDIO............................1

INTRODUCTION 1
SSMS STARTUP1..........1
OBJECT EXPLORER DETAILS – LESSER KNOWN GEMS 3
CONCLUSION..........7

SECTION TWO : BLOCKING AND LOCKING TROUBLESHOOTING..........8

INTRODUCTION 8
ACID INTRO..........8
LOCKING TYPE BASICS 8
BLOCKING BEHAVIOURS 9
UNDERSTANDING BLOCKING AND WAITS 10
KNOWING YOUR DEADLOCKS 11
TOOLS TO IDENTIFY BLOCKING..........12
CONCLUSION..........17

SECTION THREE : DEMYSTIFYING DEBUGGING TECHNIQUES WITH SQL SERVER..........18

INTRODUCTION 18
SHOW ME THE CODE!!! 18
DEBUGGER – GETTING STARTED..........19
WATCHING FOR VALUES 20
ADVANCED DEBUGGING OPTIONS 21
USEFUL DEBUGGER SHORTCUTS..........24
CONCLUSION..........24

SECTION FOUR : NUTS AND BOLTS OF PERMISSIONS AND SECURITY.......... 25

INTRODUCTION 25
PRINCIPALS 25
LOGINS AND USERS.......... 27
SPECIAL USER *SYSADMIN*..........29
CONCLUSION..........30

SECTION FIVE : SQL SERVER SECURITY PRACTICES 31

INTRODUCTION 31
AUDIT LOGINS.......... 31
AUTHENTICATION AND DISABLE SA 32
VIEW DATABASE PERMISSION..........33
SYSADMINS HAVE SUPER POWERS..........33
HOW TO PROTECT FROM SYSADMINS? 34
CONCLUSION.......... 37

ABOUT THE AUTHOR..........37

SECTION ONE
ZERO TO HERO WITH SQL SERVER MANAGEMENT STUDIO

INTRODUCTION

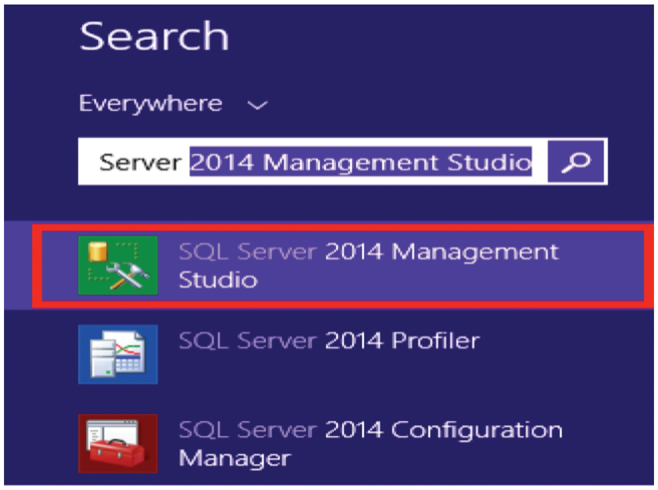
The world is full of interesting challenges and we are constantly running every single day chasing our dreams. Especially in the software ecosystem, it is not about literally our dreams but about the release scheduled in the next couple of months. We are on a constant run every single day and we stop appreciating the journey we are taking.

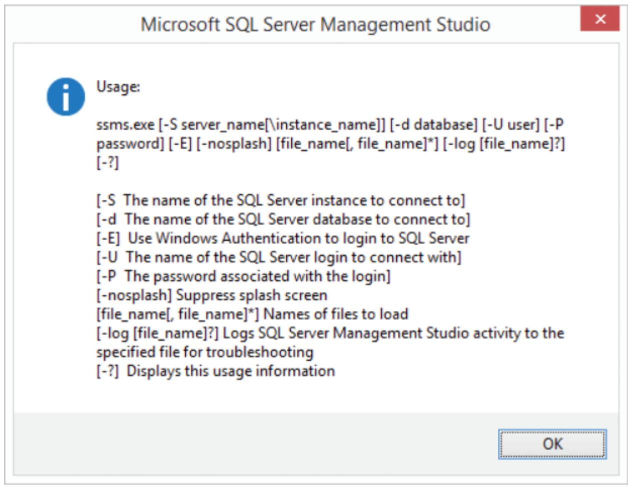
I am reminded of an interesting conversation I had with a friend few months back. In a casual visit with family, I saw his cousins had come over from out of country and they seemed to be really interested in a number local attractions in their short visit. I took note of a few names that came out in the interactions and promised my daughter to take her there soon. The whole incident made me think in a different way. Being almost in the city for more than 5 years, I am a long way from exploring what the city has to offer. Our lives are always in the fast lane and as mentioned before, we forget to enjoy and relish the journey. This analogy has so much to do with this whitepaper.

As a developer or DBA, we have our share of time working with SQL Server. In this process of working with SQL Server, the default tool of our choice is always SQL Server Management Studio. Similar to the incident shared before, we spend so much time on this fun tool that we hardly appreciate or explore what the tool has to offer. In this whitepaper, we will look at some of the capabilities available with SQL Server Management Studio that will make each one of us more productive.

SSMS STARTUP

The most common way to invoke SSMS is using the shortcut from our start menu. If you are on Windows 8 machine, then we can search for SQL Server Management Studio to invoke the program.


Search
Everywhere
Server 2014 Management Studio
SQL Server 2014 Management Studio
SQL Server 2014 Profiler
SQL Server 2014 Configuration Manager


Microsoft SQL Server Management Studio

Usage:

```
ssms.exe [-S server_name[\instance_name]] [-d database] [-U user] [-P
password] [-E] [-nosplash] [file_name[, file_name]*] [-log [file_name]?]
[-?]

[-S  The name of the SQL Server instance to connect to]
[-d  The name of the SQL Server database to connect to]
[-E]  Use Windows Authentication to login to SQL Server
[-U  The name of the SQL Server login to connect with]
[-P  The password associated with the login]
[-nosplash] Suppress splash screen
[file_name[, file_name]*] Names of files to load
[-log [file_name]?] Logs SQL Server Management Studio activity to the
specified file for troubleshooting
[-?]  Displays this usage information
```

There is yet another method to invoke SSMS by using the command prompt. In fact, the shortcut is ***SSMS***.

```
C:\> ssms
```

This will bring up the SSMS just like the shortcut we invoked before. The more interesting way to look at this shortcut is to check the various options available with SSMS command line parameters. To check them, use the following command:

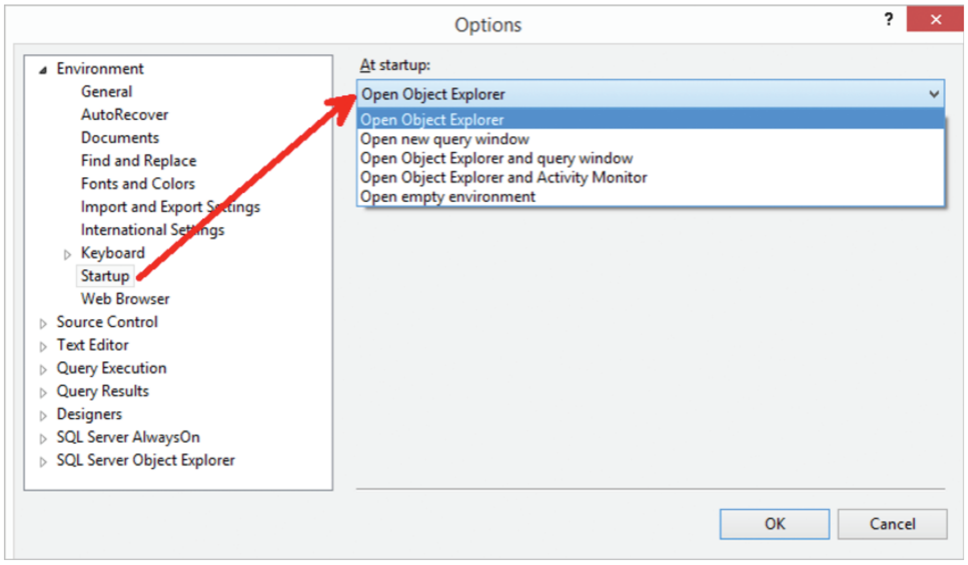
```
C:\> ssms /?
```

This bring us the various options as shown. The most interesting option here is the capability to use the –*S* option like SQL Server Name and the steps it normally takes to make an SSMS studio completely operational for work would include close to 3-4 clicks. If we use the –*E* option, then we can load the SQL Server Management Studio directly with a query window. This is the simplest way to get productive in one go.

```
Invoke SSMS with Integrated Authentication to the default server.
C:\> ssms -E

Invoke SSMS using Server name, UserName and Pwd.
C:\> ssms –S BigSQL –U Pinal –P C0mp!exPwd
```

Another extension to this tip is the ability to change how the SSMS Window starts. If we check the **Tools** → **Options** Dialog box we can see:



At startup:
Open Object Explorer

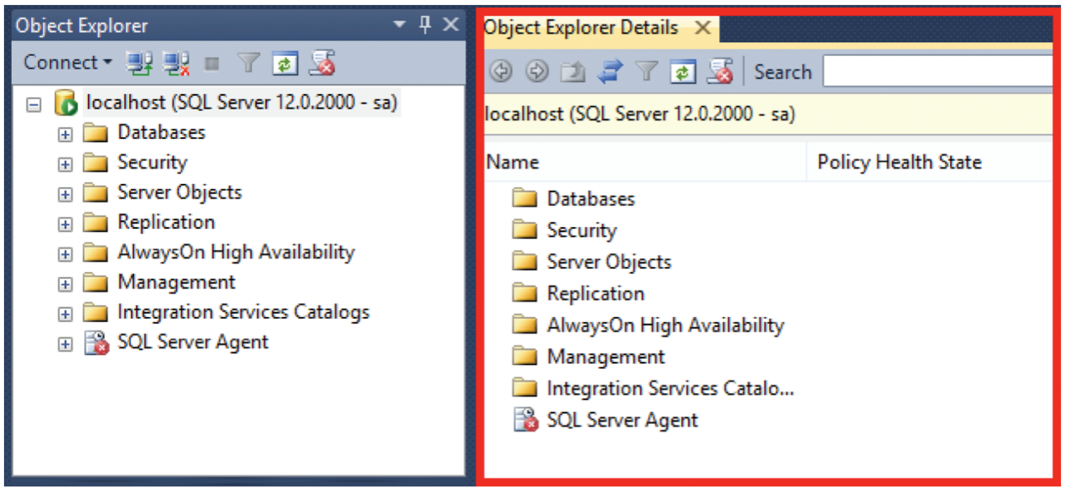
- Open Object Explorer
- Open new query window
- Open Object Explorer and query window
- Open Object Explorer and Activity Monitor
- Open empty environment

There are a number of options to choose from and the default is **Open Object Explorer**. We have the choice to change to any of the other values, and this will take effect next time we call SSMS.

OBJECT EXPLORER DETAILS – LESSER KNOWN GEMS

The best way to learn about the various capabilities of **Object Explorer Details** is by using them daily at your work. But how can one use them, if we are not sure of their capabilities? I have personally seen this Explorer as a productivity enabler.

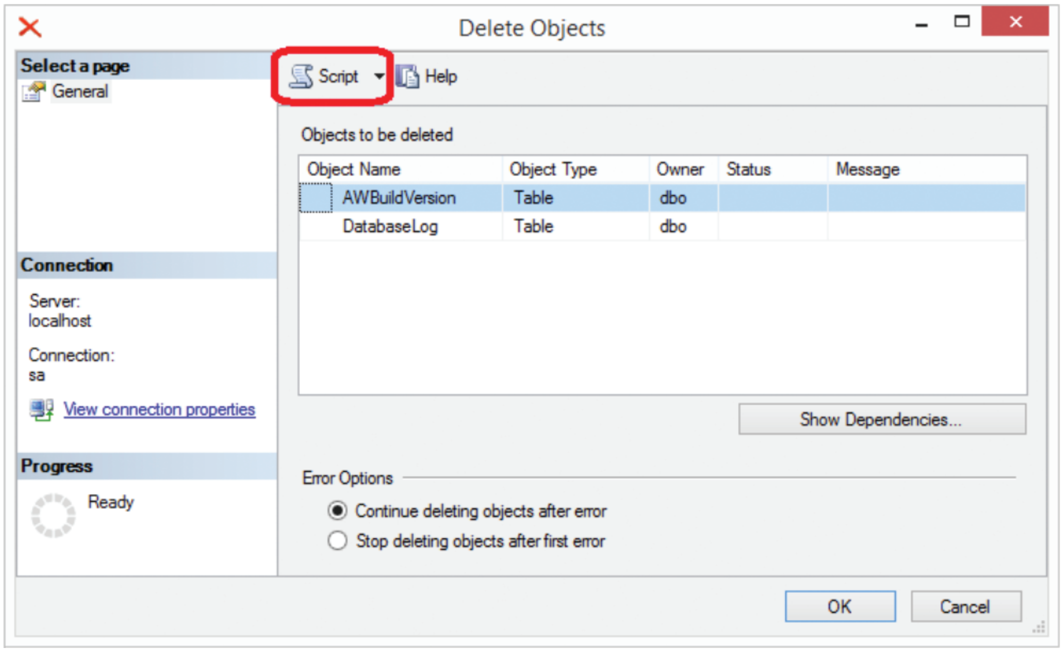
To bring up the **Object Explorer Details** pane, use the **F7** shortcut or it can be accessed from **Toolbar** → **View** → **Object Explorer Details**. Whenever I talk about the Details pane, developers sometimes get confused with the standard **Object Explorer** pane that they are used to in general.



DELETE MULTIPLE OBJECTS

Personally, I have seen this as a neat trick to ask at the local user group meetups every now and then. The question I ask is, “Is there a way to drop multiple objects in one key stroke?”. And the bonus trick question is, “Is there a way to script drop of multiple objects in one key stroke?” Either way, the simple answer to this is with **Object Explorer Details**.

To achieve this, get to the correct folder from your **Object Explorer** – let us assume this to be the **Table** node. From the **Object Explorer Details** pane, select the two objects and press the **DELETE** key. We will be presented with a dialog to delete and from the top toolbar, we can also script them out easily.



Objects to be deleted

Object Name	Object Type	Owner	Status	Message
AWBuildVersion	Table	dbo		
DatabaseLog	Table	dbo		

Connection

Server:
localhost

Connection:
sa

View connection properties

Progress

Ready

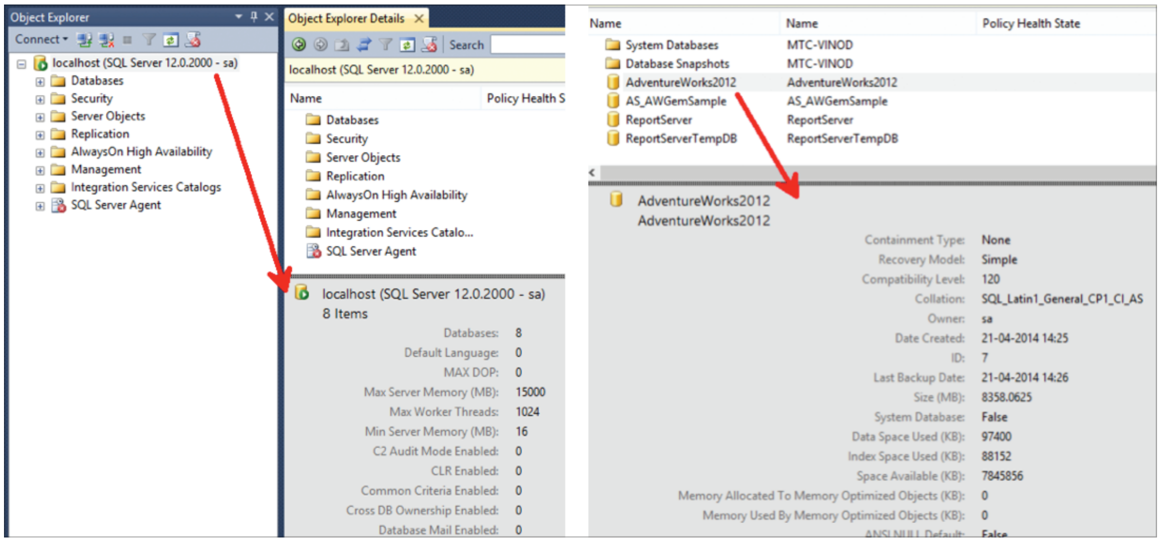
Error Options

[✓ Continue deleting objects after error]
Stop deleting objects after first error

PROPERTIES FOR OBJECTS / NODE

Depending on where the **Object Explorer Details** node is, we can get some interesting and additional information which might take us a lot of time. Here are a couple of these details which I thought would be valuable to share.

The next two images show us completely different data because the first one is around Server Level information and the second at an individual DB level.



localhost (SQL Server 12.0.2000 - sa)
8 Items

Databases:	8
Default Language:	0
MAX DOP:	0
Max Server Memory (MB):	15000
Max Worker Threads:	1024
Min Server Memory (MB):	16
C2 Audit Mode Enabled:	0
CLR Enabled:	0
Common Criteria Enabled:	0
Cross DB Ownership Enabled:	0
Database Mail Enabled:	0

Name	Name	Policy Health State
System Databases	MTC-VINOD	
Database Snapshots	MTC-VINOD	
AdventureWorks2012	AdventureWorks2012	
AS_AWGemSample	AS_AWGemSample	
ReportServer	ReportServer	
ReportServerTempDB	ReportServerTempDB	

AdventureWorks2012
AdventureWorks2012

Containment Type:	None
Recovery Model:	Simple
Compatibility Level:	120
Collation:	SQL_Latin1_General_CP1_CI_AS
Owner:	sa
Date Created:	21-04-2014 14:25
ID:	7
Last Backup Date:	21-04-2014 14:26
Size (MB):	8358.0625
System Database:	False
Data Space Used (KB):	97400
Index Space Used (KB):	88152
Space Available (KB):	7845856
Memory Allocated To Memory Optimized Objects (KB):	0
Memory Used By Memory Optimized Objects (KB):	0
ANSI NULL Default:	False

All of these data are available in DMVs. But as a DBA, having these handy in a single click via SQL Server Management Studio is really powerful. Think about questions like:

- How much RAM is on the Server?
- How many Databases are available on a given instance of Server?
- What is the Data Space used by each Database?
- How much free space is available on a given Database?
- When was the last backup taken for a given Database?
- What are the recovery models and compatibility levels for current Databases?

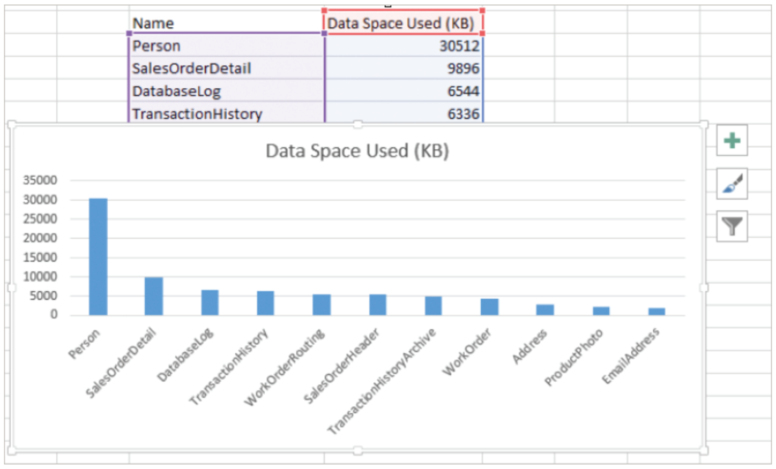
These are simple yet powerful answers we can get quickly using the details pane.

EXPLORING COPY DATA CAPABILITY

Think about a requirement to beautifully represent data about our database in a graphical form. For example, we want to build a graph of top 10 tables by size within our database. This can be easily achieved using the following steps:

1. Get to the Databases and then **Tables** folder in **Object Explorer Details**.
2. Right click the top ribbon and add column – **Data Space Used (KB)** to the details pane.
3. Remove other columns not of interest from the **Object Explorer Details** just like how we added a column in the previous step.
4. Now click on this column to sort data in descending order.
5. Next select the top 10 or how many rows as desired.
6. Press ***Ctrl + C*** to copy the contents into the **Clipboard**. If you were wondering, yes – this does work.
7. Open Excel and paste the data.
8. Press ***Alt+F1*** to build a graph.

Shown is an output of the above steps from my AdventureWorks Database.



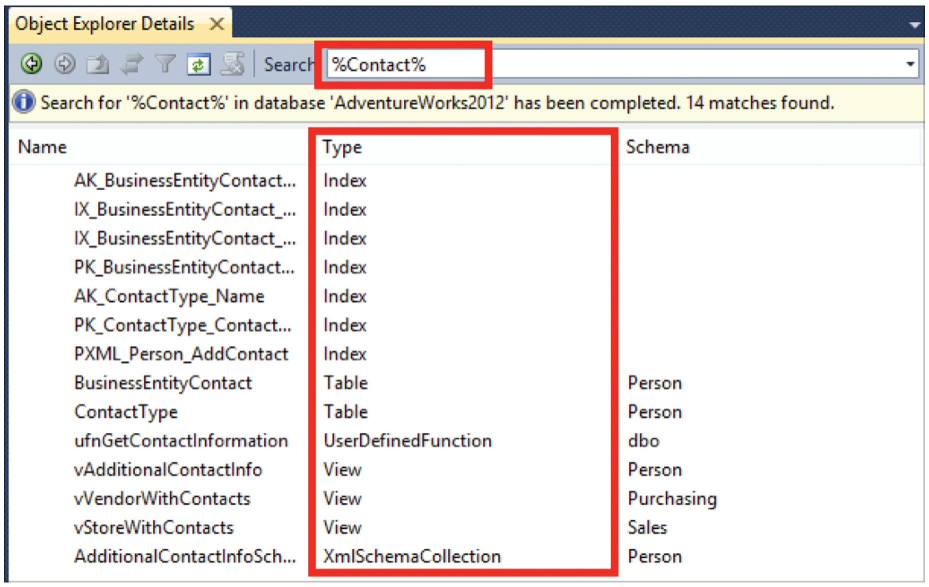
Name	Data Space Used (KB)
Person	30512
SalesOrderDetail	9896
DatabaseLog	6544
TransactionHistory	6336

TYPE AND NAVIGATE

I have had the opportunity to work on databases that have several 1000s of tables and it is sometimes difficult to get to the object from the **Object Explorer** and **Object Explorer Details** because we need to scroll through tons of data. Lesser known is the trick that we can type the object name and both **Object Explorer** and **Object Explorer Details** can take us to that point directly.

In this example, we can either type from the **Tables** node under **Object Explorer** or we can select the **Object Explorer Details** pane and type the object name. The end result for both is the same: we will automatically get scrolled to the point where the object is. The only difference being, in **Object Explorer** we need to also type the Schema name followed by the object, while in the Details pane we can directly type the Object name. Neat, isn't it?

SEARCHING OBJECTS FROM THE WHOLE DB



Search for '%Contact%' in database 'AdventureWorks2012' has been completed. 14 matches found.

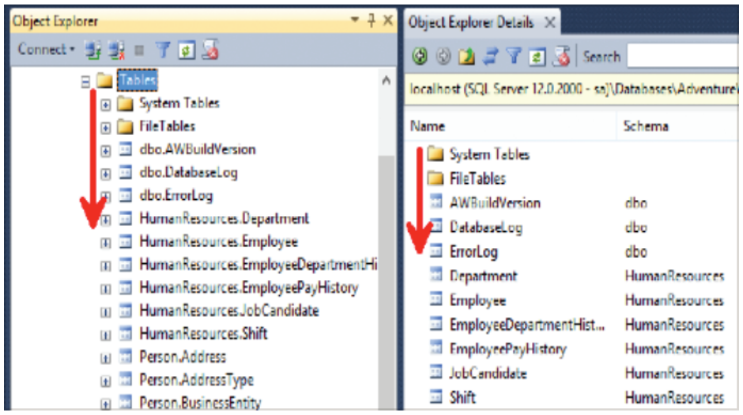
Name	Type	Schema
AK_BusinessEntityContact...	Index	
IX_BusinessEntityContact_...	Index	
IX_BusinessEntityContact_...	Index	
PK_BusinessEntityContact...	Index	
AK_ContactType_Name	Index	
PK_ContactType_Contact...	Index	
PXML_Person_AddContact	Index	
BusinessEntityContact	Table	Person
ContactType	Table	Person
ufnGetContactInformation	UserDefinedFunction	dbo
vAdditionalContactInfo	View	Person
vVendorWithContacts	View	Purchasing
vStoreWithContacts	View	Sales
AdditionalContactInfoSch...	XmlSchemaCollection	Person

Let us take a scenario where we need to search for a keyword and search across our database for objects having this keyword. Now, we need to search across object types like Tables, Stored Procedures, Triggers, Synonyms, etc. This can be achieved using the **Object Explorer Details** pane. There is a search bar right on the top and type in the keywords to find them.

In this example, we have searched for the keyword "Contact" and we are using "%" as a wildcard character. This tells SQL Server Management Studio to display all the objects that have the word "Contact" in their name.

As we can see from the **Type** column, we have a mix of Indexes, Table, userDefinedFunctions, Views and XML Schema Collection as part of the search we just performed. The top bar shows interesting information – it says "Search for '%Contact%' in database 'AdventureWorks2012' has been completed."

If you want, we can expand on this search capability and do the same search at an instance level. If we select the Server name under **Object Explorer** and perform a search here with a wildcard, this search will happen across databases, across all objects. As you can see, the message is slightly different when compared to what we saw previously.



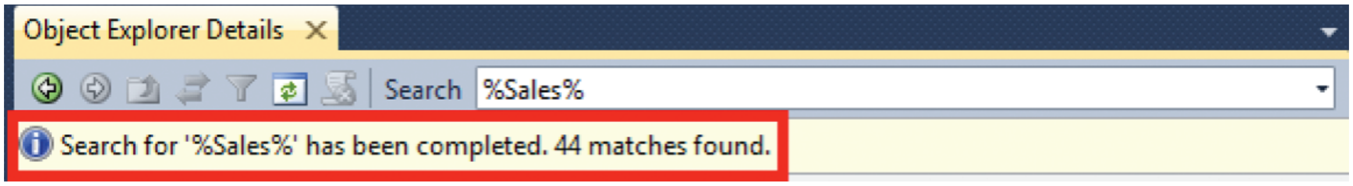
Object Explorer

Tables
- System Tables
- FileTables
- dbo.AWBuildVersion
- dbo.DatabaseLog
- dbo.ErrorLog
- HumanResources.Department
- HumanResources.Employee
- HumanResources.EmployeeDepartmentHi
- HumanResources.EmployeePayHistory
- HumanResources.JobCandidate
- HumanResources.Shift
- Person.Address
- Person.AddressType
- Person.BusinessEntity

Object Explorer Details

localhost (SQL Server 12.0.2000 - sa)\Databases\Adventure

Name	Schema
System Tables	
FileTables	
AWBuildVersion	dbo
DatabaseLog	dbo
ErrorLog	dbo
Department	HumanResources
Employee	HumanResources
EmployeeDepartmentHist...	HumanResources
EmployeePayHistory	HumanResources
JobCandidate	HumanResources
Shift	HumanResources



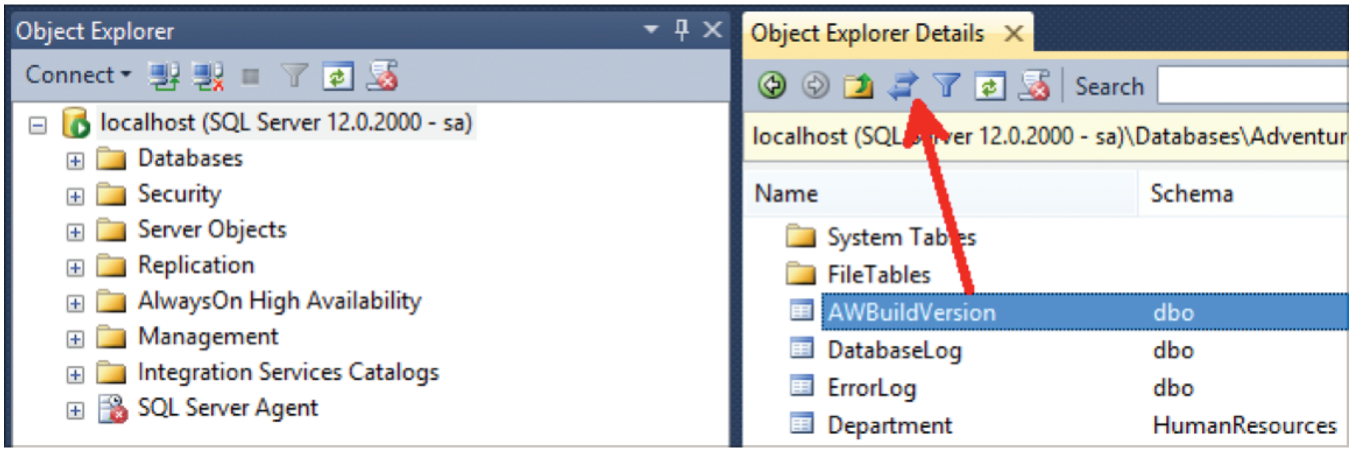
Object Explorer Details

Search %Sales%

Search for '%Sales%' has been completed. 44 matches found.

SYNC WITH OBJECT EXPLORER DETAILS

A lot of times when we start our work using **Object Explorer Details**, we can be working inside a shell that is not in sync with the **Object Explorer** pane. Sometimes we would love to get to that node on our **Object Explorer** pane, and this can be easily achieved using the **Sync** button on top of the **Object Explorer Details** pane.



Object Explorer

localhost (SQL Server 12.0.2000 - sa)
- Databases
- Security
- Server Objects
- Replication
- AlwaysOn High Availability
- Management
- Integration Services Catalogs
- SQL Server Agent

Object Explorer Details

localhost (SQL Server 12.0.2000 - sa)\Databases\Adventur

Name	Schema
System Tables	
FileTables	
AWBuildVersion	dbo
DatabaseLog	dbo
ErrorLog	dbo
Department	HumanResources

KNOW YOUR FILTERS

If you work with enterprise level software, the chances of you dealing with 100s of tables if not 1000s is quite possible. I have seen developers and DBAs struggle with scrolling through tons of data. Earlier in this whitepaper we showed you ways to search, and type to get to the location. There is yet another method to work with the data, which is using Filters.

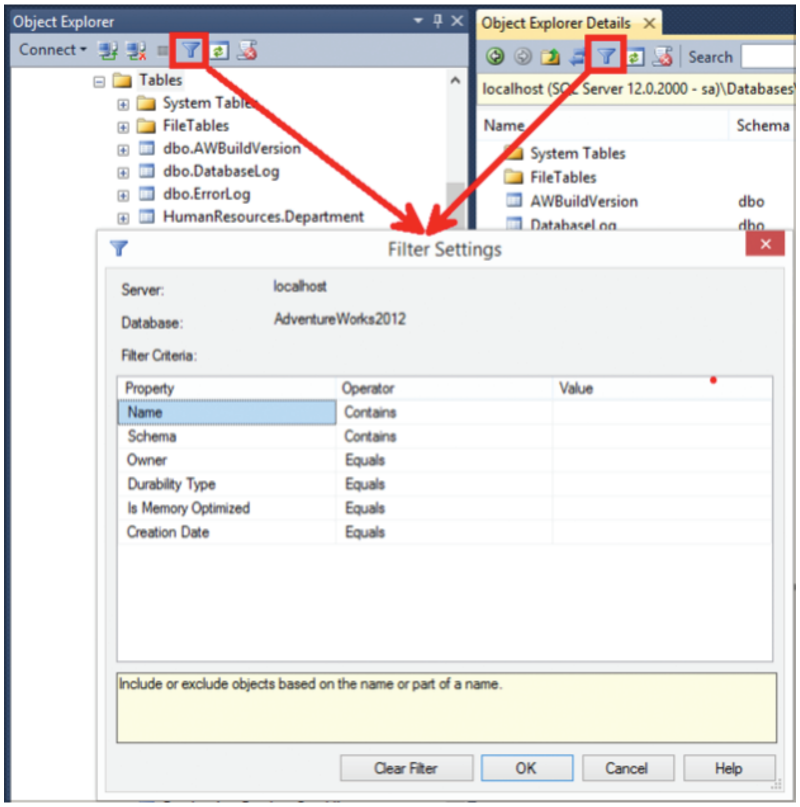
There are a couple of ways we can do this.

1. Use the Object Explorer to get to the given node and in the above example we are at the **Tables** node and we have used the **Filter** icon available in Object Explorer.

2. We can get the same effect of filter and the dialog from our **Object Explorer Details** pane. Similar to the earlier point, drill down to the node under question and then click on the **Filter** icon from the toolbar.

Adding a filter for an object like 'Person', 'Sales' or anything else will filter both on the **Object Explorer** pane and **Object Explorer Details** pane at the same time.

I highly recommend using this technique because we do not have to scroll through tons of data before narrowing down to our desired content.



Server: localhost

Database: AdventureWorks2012

Filter Criteria:

Property	Operator	Value
Name	Contains	
Schema	Contains	
Owner	Equals	
Durability Type	Equals	
Is Memory Optimized	Equals	
Creation Date	Equals	

Include or exclude objects based on the name or part of a name.

CONCLUSION

Personal productivity is one of the key things a developer or a DBA loves to know about. In this whitepaper, I have covered some of the simplest of tricks using **Object Explorer** and **Object Explorer Details** that you can start using in your environment. These are simple yet powerful shortcuts that each one of us can use effectively and immediately for our work.

SECTION TWO
BLOCKING AND LOCKING TROUBLESHOOTING

INTRODUCTION

How many times in your life have you had to stand in a long queue to get work done? Be it the railway station, waiting for a bus or anything else? Why do we all do this? There is a process and we need to follow the rules to get things done. If you look at it closely, there is a method to the madness and to avoid any chaos we need to follow these rules. If I had to compare this to SQL Server world there are tons of similarities. To maintain orderliness one has to stand in a queue, the longer the person in front takes to complete their task, the longer we will be waiting in the queue. If this were a railway station ticketing counter, then if the person takes a long time at the counter, the ticket master is unavailable to issue tickets for others. So where is the similarity? Well, waiting in a queue while the previous person finishes their task is a typical blocking behavior and the state where the ticket master is not able to issue tickets to others is a classic locking of resource problem. This whitepaper is all about blocking and locking inside SQL Server.

ACID INTRO

Let me take you through the same from the basics. Blocking and locking is inevitable in traditional relational databases. Like in our example above, it is a process and to ensure ACID properties of transactions, one needs to have them.

Atomicity: Data modifications in a transaction is all-in or none behaviour.

Consistency: Once a transaction is committed, data must be in a consistent state, i.e. data integrity needs to be met.

Isolation: This is isolating and protecting concurrent transactions in modifying data that have been changed by another concurrent transaction.

Durability: As the name suggests, once a transaction completes, the modifications are permanent and persist even in event of system failures.

Now that we are aware of the basics, SQL Server uses locks on data to prevent data corruption and stop multiple users from updating the same record at the same time.

LOCKING TYPE BASICS

Let me take a moment to talk only about the most commonly used locking types in this section. Though this is a complex and heavy topic, it is helpful to get a primer in this section. There are a number of lock types, but the most common ones are:

Shared: This is used to allow concurrent transactions to read source data. The shared (*S*) lock type is released as soon as the data has been read, unless the Isolation level is repeatable read or higher.

Exclusive: This lock is used to make sure no other transactions can read or modify data locked with an exclusive (*X*) lock.

Intent: This lock type indicates that SQL Server wants to acquire a shared (*S*) lock or exclusive (*X*) lock on some of the resources lower down in the transaction process.

Update: The update (*U*) lock is used to prevent deadlock as exclusive from being used until a modification is made. A typical update would acquire a shared (*S*) lock on the resources and then modifying would require the locks to be converted to exclusive (*X*) locks. If two transactions try to perform an update data while one data tries to convert into exclusive lock, this transaction needs to wait as the shared lock from other transaction and this conversion to exclusive lock are not compatible. If the second transaction also tries to convert into an exclusive lock, then this transaction is waiting for connection one to release its lock. This is a typical scenario of cyclic deadlock which we will explain in detail later. To avoid this scenario, update locks are used by SQL Server because only one transaction can hold an Update lock on the same resource at a point in time.

In the below table, I have outlined a few more locking types for reference.

Lock	Description
Schema-Stability (*Sch-S*)	Acquired when compiling queries
Schema Modification (*Sch-M*)	Acquired for DDL operations like **ALTER** or **DROP** on a table schema
Shared (*S*)	Acquired for reading data
Update (*U*)	Acquired before modification can get converted to exclusive
Exclusive (*X*)	Acquired for writing
Intent Shared (*IS*)	Requests for shared lock(s)
Intent Update (*IU*)	Requests for update lock(s)
Intent Exclusive (*IX*)	Requests for exclusive lock(s)
Bulk Update (*BU*)	Used when we do bulk copy operations into a table

Locks on a resource can be taken for a short or a long duration. Short locks are released before the transaction completes. These are like Shared lock in Read Committed Isolation wherein the lock is released as soon as the transaction completes. Long locks are those where the locks are released only when the transaction completes. Typically these are like exclusive locks taken for insert, update or delete of rows. It is these uncommitted transactions that hold onto locks and cause possible blocking behavior for other connections.

Note: Locks can also be held during sorting or hashing of rows as the query is waiting for memory resources.

Yet another reason can be because of IO intensive queries. I wrote a complete series on wait stats and will refrain from expanding on them here again.

One salient point to note is that locks are managed on a per connection basis.

If we are talking about locking so much, what is blocking then? Let us briefly discuss this topic.

BLOCKING BEHAVIOURS

Blocking is a scenario where two connections are fighting over an incompatibility lock on a resource, i.e., table, row, page, ranges of keys, indexes or database. This is a first-come first-serve basis scenario. The first connection that makes a request for a lock is granted access to the resource while all the subsequent requests are now blocked and cannot continue processing until the first connection's lock is released. By default, there is no mandatory time-out period and no way to test if a resource is locked before locking it, except to attempt to access the data (and potentially get blocked indefinitely). Blocking, as you can see, is inevitable and is needed for data integrity (from ACID it is consistency and isolation). It is surely an extension to the locking basics we discussed before in this whitepaper.

UNDERSTANDING BLOCKING AND WAITS

Since we are talking about waits at the row / page level, this information can be gotten from the **Dynamic Management View (DMV)** - *SYS.DM_DB_INDEX_OPERATIONAL_STATS* using the following query. The query finds the blocking and the wait times for the given database.

```
-- Calculate the blocking rates and wait times
SELECT SUM(row_lock_count) row_locks,
SUM(row_lock_wait_count) row_lock_waits,
SUM(row_lock_wait_in_ms) row_lock_wait_time_ms,
SUM(page_lock_count) page_locks,
SUM(page_lock_wait_count) page_lock_waits,
SUM(page_lock_wait_in_ms) page_lock_wait_time_ms
FROM sys.dm_db_index_operational_stats(DB_ID(),null,null,null)
```

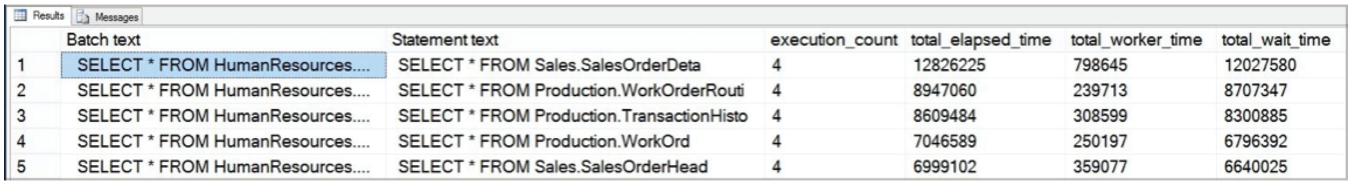
These DMVs are so powerful that we can get tons of other information to how the access pattern has been on these resources. For example, we can be interested in understanding the number of inserts, updates and deletes happening on the database. This can be also queried from the same DMV using a query as shown below:

```
SELECT DB_NAME() as DB_NAME, obj.name as table_name,
ind.name as index_name, ind.type_desc,
leaf_allocation_count+nonleaf_allocation_count as splits,
range_scan_count, singleton_lookup_count,
leaf_insert_count+nonleaf_insert_count as inserts,
leaf_update_count+nonleaf_update_count as updates,
leaf_delete_count+nonleaf_delete_count as deletes
FROM sys.dm_db_index_operational_stats(DB_ID(),null,null,null) as os
INNER JOIN sys.indexes as ind
ON ind.object_id = os.object_id and ind.index_id = os.index_id
INNER JOIN sys.objects as obj
ON obj.object_id = os.object_id
WHERE obj.Type NOT LIKE 'S'
```

Let us query *SYS.DM_EXEC_QUERY_STATS* and *SYS.DM_EXEC_SQL_TEXT* to identify the top 20 blocked queries and their wait times using the below query.

```
SELECT TOP(20) sql_text.text AS "Batch text",
CASE qs.statement_end_offset
     WHEN -1
     THEN SUBSTRING(sql_text.text, qs.statement_start_offset/2,64000)
     ELSE SUBSTRING(sql_text.text, qs.statement_start_offset/2,
            (qs.statement_end_offset-qs.statement_start_offset)/2)
     END AS "Statement text",
execution_count, total_elapsed_time, total_worker_time,
(total_elapsed_time - total_worker_time) AS total_wait_time
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS sql_text
ORDER BY (total_elapsed_time - total_worker_time) DESC
```

A typical output looks like this:



	Batch text	Statement text	execution_count	total_elapsed_time	total_worker_time	total_wait_time
1	SELECT * FROM HumanResources....	SELECT * FROM Sales.SalesOrderDeta	4	12826225	798645	12027580
2	SELECT * FROM HumanResources....	SELECT * FROM Production.WorkOrderRouti	4	8947060	239713	8707347
3	SELECT * FROM HumanResources....	SELECT * FROM Production.TransactionHisto	4	8609484	308599	8300885
4	SELECT * FROM HumanResources....	SELECT * FROM Production.WorkOrd	4	7046589	250197	6796392
5	SELECT * FROM HumanResources....	SELECT * FROM Sales.SalesOrderHead	4	6999102	359077	6640025

KNOWING YOUR DEADLOCKS

Before we can get into a typical blocking behavior and troubleshooting, let us take a moment to recognize a unique behavior called as deadlocks. A typical deadlock is a case when two connections are waiting to release resources locked by the other connection. A common form of deadlock is called a cyclic deadlock. To outline how a typical cyclic deadlock happens, check the sequence of activity happening in two connections.

Time	Connection 1	Connection 2
T1	**Begin Tran**	**Begin Tran**
T2 Granted	**Update Persons** **Set DOJ = "10/01/2014'** **Where name like 'Pinal'**	
T3 Granted		**Update Address** **Set active = 'N'** **Where person_name like 'Pinal'**
T4 Request	**Select*** **From Address** **Where person_name like 'Pinal'**	
T5 Request		**Select*** **From Persons** **Where name like 'Pinal'**
T6	**Deadlock Victim**	**(blocking removed)**
T7		Commit

In the example, we can see that two connections are trying to take a lock on “Persons” and “Addresses” tables within the same connection timespan. The sequence is made in such a way that now each of the connections is waiting for release of locks from other connection. We are sure, if you ever encountered this deadlock situation – then a typical error message is shown:

```
Msg 1205, Level 13, State 56, Line 10
Transaction (Process ID 53) was deadlocked on lock resources
with another process and has been chosen as the deadlock victim.
Rerun the transaction.
```

Though deadlocks happen, not many know that a deadlock need not happen only because of cyclic behavior as explained above. It can also occur in a single resource too. Let me show this in a simple timeline.

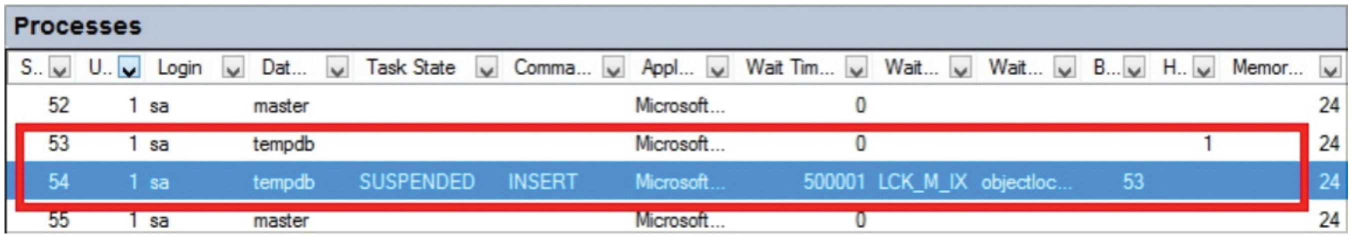
Time	Connection 1	Connection 2
T1	**Begin Tran**	**Begin Tran**
T2 Granted	**Select* From Persons With (HOLDLOCK) Where name like ‘Pinal’**	
T3 Granted		**Select* From Persons With (HOLDLOCK) Where name like ‘Pinal’**
T4 Blocked	**Update Persons Set active = ‘N’ Where name like ‘Pinal’**	
T5 Blocked		**Update Persons Set active = ‘Y’ Where name like ‘Pinal’**
T6	**Deadlock Victim**	**(blocking removed)**
T7		Commit

TOOLS TO IDENTIFY BLOCKING

There are a number of tools available with SQL Server that are out-of-the-box which we can use to identify blocking behavior. There is no one tool that will fit the bill for all blocking issues. We need to use the right tool for the right situation.

ACTIVITY MONITOR

One of the hidden gems inside SQL Server Management Studio is the **Activity Monitor** pane. The shortcut to invoke this window is “**CTRL + ALT + A**”. From a blocking point of view, this can show us which are the processes blocked and who is blocking with the resource associated. In the below screenshot, I have shown a simple blocking behaviour.



Processes

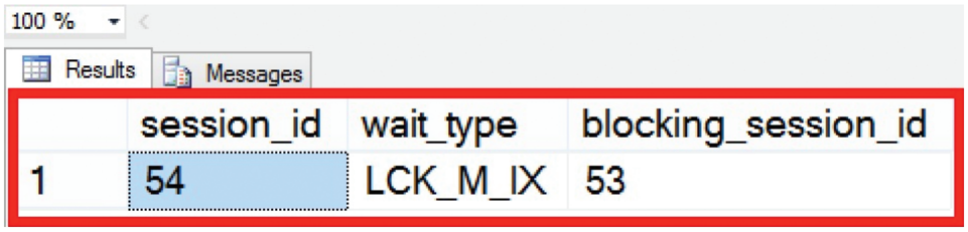
S..	U..	Login	Dat...	Task State	Comma...	Appl...	Wait Tim...	Wait...	Wait...	B...	H..	Memor...
52	1	sa	master			Microsoft...	0					24
53	1	sa	tempdb			Microsoft...	0				1	24
54	1	sa	tempdb	SUSPENDED	INSERT	Microsoft...	500001	LCK_M_IX	objectloc...	53		24
55	1	sa	master			Microsoft...	0					24

The way to read this is simple. Look for the value of 1 in the "Head Blocker" field; in our example it is SPID of 53. Next look for values in "Blocked By"; in our example above, the **Blocked By** column has 53, which means the SPID of 54 is being blocked by SPID of 53. Hence on current live systems it is worth to note that this is the fastest and quickest way to find blocking queries on a live system to start troubleshooting. This is very basic information to start, but more often we want a lot more information about the blocking behavior which can be gotten by tons of DMVs available inside SQL Server. Next we will look at some of them.

DMVs

The **Dynamic Management Views (DMVs)** can be defined as a set of predefined views given out-of-the-box by SQL Server to understand, monitor and troubleshoot activities happening inside SQL Server. With each release of SQL Server, the number of DMVs keeps increasing because we have additional features to monitor. Getting back to blocking, we can easily get information.

```
USE MASTER
GO
SELECT session_id, wait_type, blocking_session_id
FROM sys.dm_os_waiting_tasks
WHERE blocking_session_id <> 0
GO
```



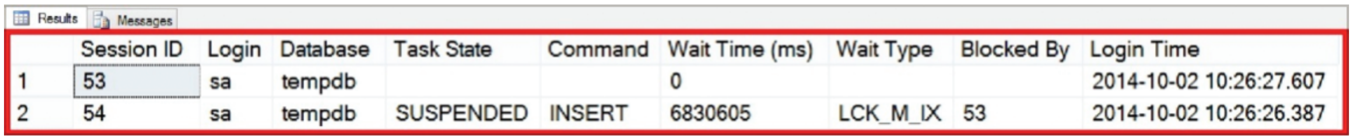
	session_id	wait_type	blocking_session_id
1	54	LCK_M_IX	53

A more complex query with tons of additional fields can be gotten from multiple DMVs. This is something I shared over my blog and thought is worth a note here for quick reference.

```
SELECT
    [Session ID]          = s.session_id,
    [Login]               = s.login_name,
    [Database]            = case when p.dbid=0 then N''
                                else ISNULL(db_name(p.dbid),N'') end,
    [Task State]          = ISNULL(t.task_state, N''),
    [Command]             = ISNULL(r.command, N''),
    [Wait Time (ms)]      = ISNULL(w.wait_duration_ms, 0),
    [Wait Type]           = ISNULL(w.wait_type, N''),
    [Blocked By]          = ISNULL(CONVERT (varchar, w.blocking_session_id), ''),
    [Login Time]          = s.login_time
```

```
FROM sys.dm_exec_sessions s LEFT OUTER JOIN sys.dm_exec_connections c
     ON (s.session_id = c.session_id)
LEFT OUTER JOIN sys.dm_exec_requests r ON (s.session_id = r.session_id)
LEFT OUTER JOIN sys.dm_os_tasks t ON (r.session_id = t.session_id
     AND r.request_id = t.request_id)
LEFT OUTER JOIN sys.dm_os_waiting_tasks w
ON (t.task_address = w.waiting_task_address)
LEFT OUTER JOIN sys.dm_exec_requests r2
     ON (s.session_id = r2.blocking_session_id)
LEFT OUTER JOIN sys.sysprocesses p ON (s.session_id = p.spid)
WHERE s.is_user_process = 1
AND (r2.session_id IS NOT NULL
OR w.blocking_session_id IS NOT NULL)
ORDER BY s.session_id;
```

The same output is now shown with more details from multiple DMVs. This can give you a rough idea as to how powerful the DMVs are inside SQL Server.

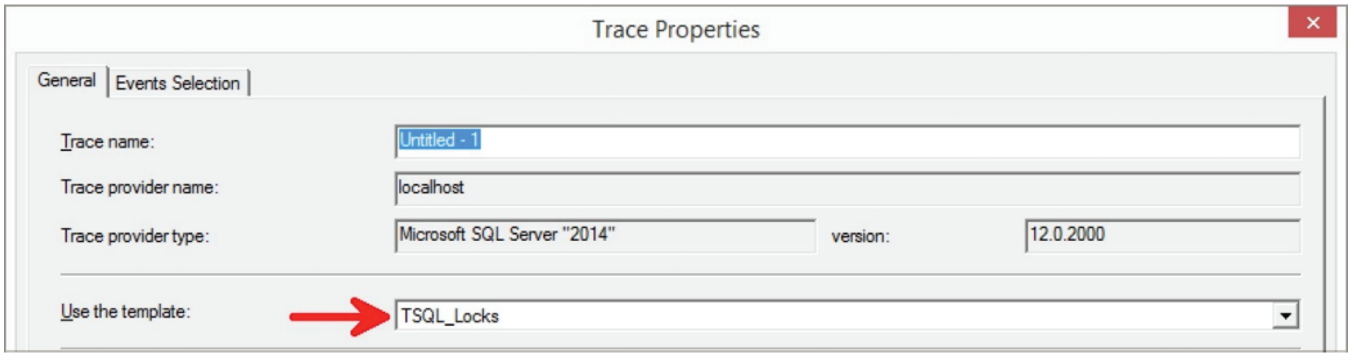

Results
Messages

	Session ID	Login	Database	Task State	Command	Wait Time (ms)	Wait Type	Blocked By	Login Time
1	53	sa	tempdb			0			2014-10-02 10:26:27.607
2	54	sa	tempdb	SUSPENDED	INSERT	6830605	LCK_M_IX	53	2014-10-02 10:26:26.387

PROFILER

Profiler has been with SQL Server for close to 1.5 decades and most users (DBAs and Developers alike) rely on this tool heavily. As a DBA, this can be an awesome tool to troubleshoot activities happening inside SQL Server.

Profiler can also be powerful in troubleshooting blocking, deadlock, waiting and more. One of the lesser known facts is the way in which we use trace templates. One of the default templates that come with profiler is called as TSQL_Locks. As the name suggests, it gives us vital information about locks that have happened inside our SQL Server instance.



Trace name: Untitled - 1
Trace provider name: localhost
Trace provider type: Microsoft SQL Server "2014" version: 12.0.2000
Use the template: TSQL_Locks

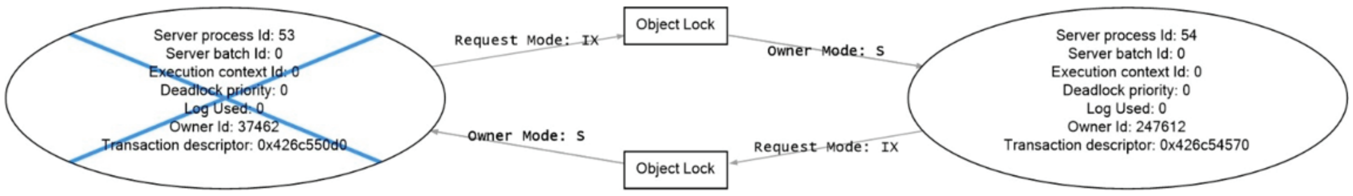
Since profiler collects specific events, the TSQL_Locks template collects the following events.

- Blocked Process Report
- SP: StmtCompleted
- SP: StmtStarting
- SQL: StmtCompleted
- SQL: StmtStarting
- Deadlock Graph
- Lock: Cancel
- Lock: Deadlock
- Lock: Deadlock Chain
- Lock: Escalation
- Lock: Timeout (timeout>0)

If you haven't used this before with your SQL Server environments, then I highly recommend giving it a try as part of your troubleshooting. Having said that, from SQL Server 2014, the profiler tool has been deprecated as we seem to move slowly but surely towards XEvents. We will discuss them later in this paper.

PROFILER – DEADLOCK GRAPHS

There are a number of events mentioned above worth a look; let me take one of the most interesting events called "Deadlock Graph". As the name suggests, it is a visual representation of how deadlock has occurred and what are the connections involved.



We just simulated a single resource deadlock and we can see how the deadlock graph looks like.

As we said, deadlocks are a special case scenario of locking. It is important to mention that from SQL Server 2005 we also had another capability called "Blocked Process Report". This is also available in the above template.

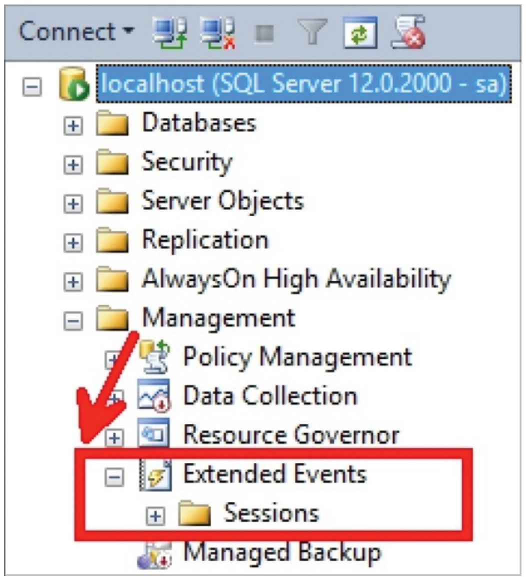
A Blocked Process Report is invoked once we configure the same. The idea here is to have a master switch which will trigger an event once a connection is waiting for a resource for more than the threshold time interval specified in the configuration.

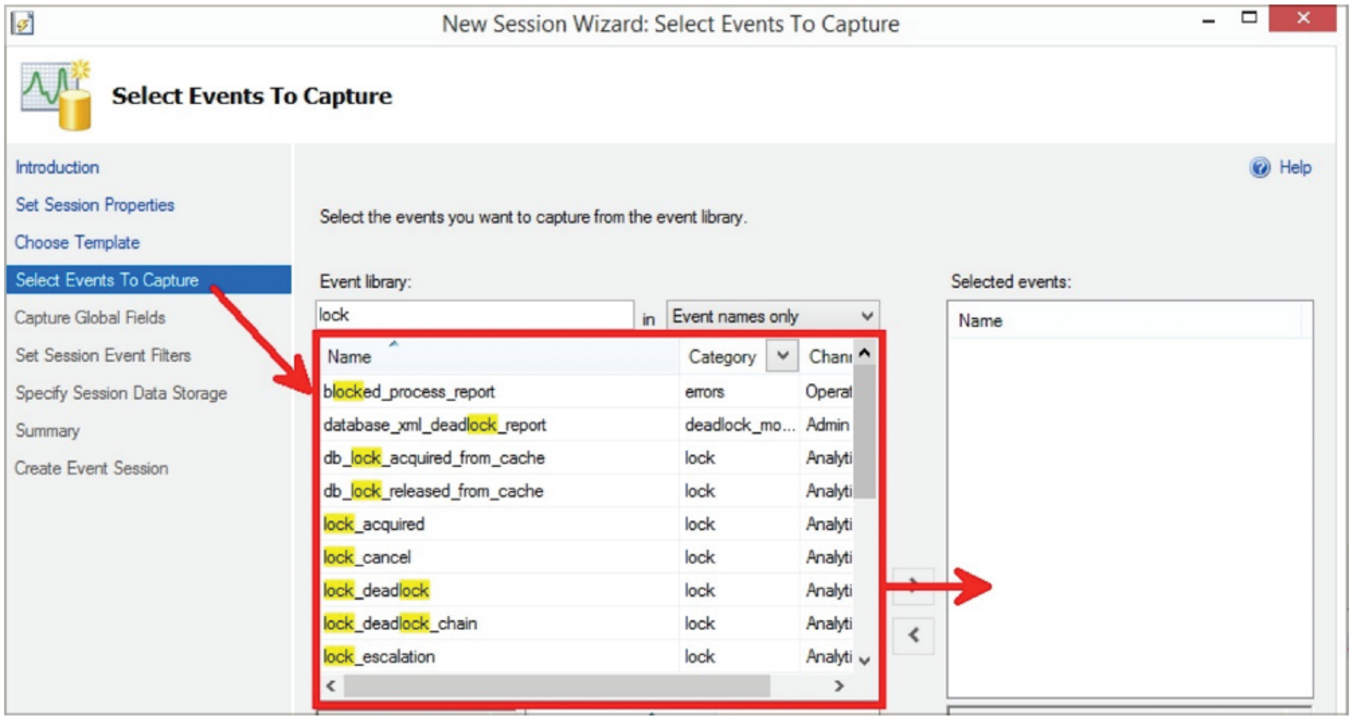
EXTENDED EVENTS

Extended Events (also called XEvents) were available from SQL Server 2005 edition, but it took prominence from SQL Server 2008 R2 release. It has evolved from a mere TSQL syntax to the latest release of SQL Server 2014, and we now have a decent UI to work with. This feature was introduced for lightweight logging and profiling – something similar to profiler but with much more capabilities. This is the main reason for Profiler now being deprecated because this is the future of troubleshooting and logging inside SQL Server.

In this release of SQL Server, we can find a new node called Extended Events and we can start a new session wizard by right clicking the **Sessions** node.

The wizard is quite self-explanatory, there are tons of events to choose from and in the below example we have gone ahead and searched on "Lock" related events. As you can see, we have almost similar events as defined in Profiler. On closer look, actually there are a lot more than what Profiler can give.





Select the events you want to capture from the event library.

Event library: lock in Event names only

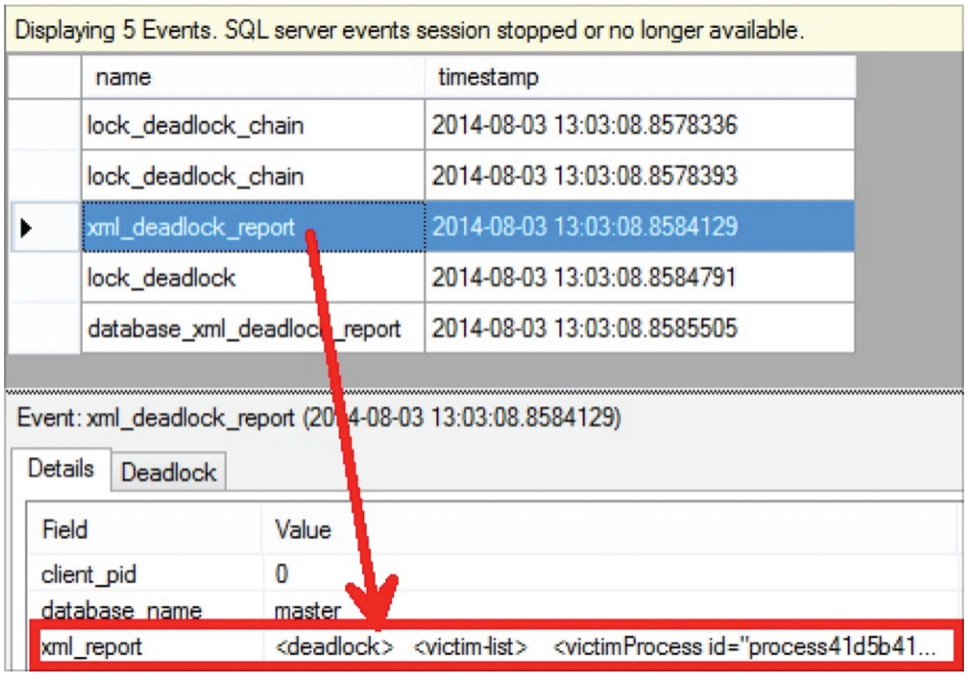
Selected events:

Name	Category	Chan
blocked_process_report	errors	Operat
database_xml_deadlock_report	deadlock_mo...	Admin
db_lock_acquired_from_cache	lock	Analyti
db_lock_released_from_cache	lock	Analyti
lock_acquired	lock	Analyti
lock_cancel	lock	Analyti
lock_deadlock	lock	Analyti
lock_deadlock_chain	lock	Analyti
lock_escalation	lock	Analyti

We have gone ahead and selected few events and clicked our way to close. At the finalize screen we have the option to script out the command that runs behind the scenes. The TSQL that gets generated for a sample XEvent I created is:

```
CREATE EVENT SESSION [Locks] ON SERVER
ADD EVENT
      sqlserver.database_xml_deadlock_report(ACTION(sqlserver.client_pid,
      sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT
      sqlserver.lock_deadlock(ACTION(sqlserver.client_pid,
      sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT
      sqlserver.lock_deadlock_chain(ACTION(sqlserver.client_pid,
      sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT
      sqlserver.xml_deadlock_report(ACTION(sqlserver.client_pid,
      sqlserver.database_name,sqlserver.sql_text))
WITH (STARTUP_STATE=ON)
GO
```

Once the session event is started, it starts to collect data. If you are on SQL Server Management Studio then we can also do a “Live Preview” of the collection made. In my example, I have gone ahead and simulated a deadlock similar to the above. And if we watch our events collection for the session, we can get the XML_DEADLOCK_REPORT, which is similar to the graphical report we got from Profiler in XML format.



Displaying 5 Events. SQL server events session stopped or no longer available.

name	timestamp
lock_deadlock_chain	2014-08-03 13:03:08.8578336
lock_deadlock_chain	2014-08-03 13:03:08.8578393
xml_deadlock_report	2014-08-03 13:03:08.8584129
lock_deadlock	2014-08-03 13:03:08.8584791
database_xml_deadlock_report	2014-08-03 13:03:08.8585505

Event: xml_deadlock_report (2014-08-03 13:03:08.8584129)

Field	Value
client_pid	0
database_name	master
xml_report	<deadlock> <victim-list> <victimProcess id="process41d5b41...

As you can see, the results and reports contain interesting information which we will never be able to collect using Profiler. Hence it makes sense, this is the future. I highly urge you to get accustomed with XEvents.

CONCLUSION

Blocking and locking inside SQL Server is part of the system. It is inevitable, because to maintain integrity of data and show consistent data to users accessing the system, locking is important. As a special case scenario, keep an eye on Deadlocks, and the best way to mitigate them will be proper coding practices. This paper talked about how to troubleshoot and identify blocking behaviour. We have not explained in detail how to mitigate the situation yet, but these quick troubleshooting techniques will surely make you efficient in looking at locks and blocking situations.

SECTION THREE
DEMYSTIFYING DEBUGGING TECHNIQUES WITH SQL SERVER

INTRODUCTION

The greatest happiness for a father is to see our children grow in front of our very own eyes. My daughter has been my source of inspiration for a number of things that I face in life. It is like living your childhood all over again with a twist. I have nothing to complain about but everything to cherish. Recently, I had a unique opportunity to go to my daughter's school for the parent-teachers meeting.

In my last visit to the school, I had an opportunity to observe almost every parent around and how they were handling their kids. There was something I noticed consistently that is worth a mention. The kids surely have a lot of energy and are filled with questions. There wasn't a minute where I would hear a kid ask – "Why, papa?", or "Why is it like this?" This was the consistent questioning to understand more about things happening around them, and they are explorers. The more I watched the whole exercise, the more I started enjoying the day with my daughter. It taught me interesting things that I think are worth sharing. If you ask me in software terms, the kid is debugging each and every result in front of them as they understand the reason.

Just the need to question why things are different and why they are behaving the way they are is a typical instinct of a Developer writing code. Many times I write some pseudo code and when I translate it into actual code, I get baffled with why the result is so different while my logic says something else. This leads me to debug the whole exercise.

As a SQL Server developer, the fundamental fun of debugging started in my early days using the PRINT command inside SQL Server. I personally get a chance to see these debugging practices still alive in lot of production code. In this whitepaper, let me take a chance to introduce you to the debugger techniques introduced with SQL Server 2012. If you are a Visual Studio C# or a .NET developer, these techniques are so similar that you will never forget them. Yes, it is important to know and note that the SQL Server Management Studio IDE is based on the Visual Studio shell. Hence the goodness of VS carries forward with Management Studio too.

SHOW ME THE CODE!!!

The idea of this paper is not to write lengthy code blocks and show you how a debugger might potentially work. We will use a simple looping block as shown below to check how a debugger will work.

The code is simple as it initializes a variable. We have a *WHILE* block which loops while incrementing the variable by 20. In every 200 values achieved, we are printing the same as part of the code. The code is proposed to run till 10000.

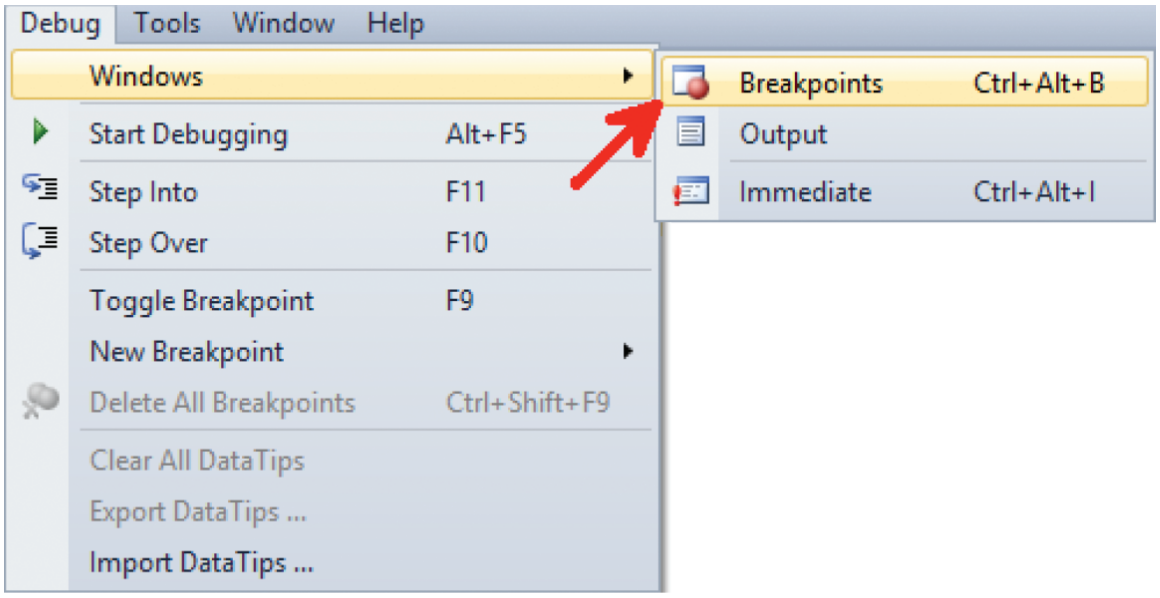
```sql
-- Simple loop for Debugger Basics
DECLARE @loop INT = 0
WHILE (@loop < 10000)
BEGIN
    IF (@loop%200 = 0)
    -- Used to track our values
    PRINT @loop

    SET @loop += 20
END
```

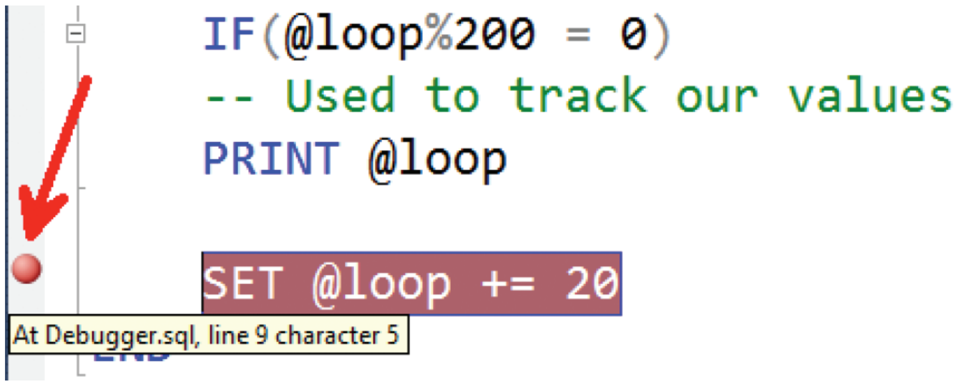
There is absolutely no rocket science to this code. Let us assume this can be code that runs in your production environment where we need to debug how the variable value increases in each of the passes.

DEBUGGER – GETTING STARTED

There are a number of ways we can start the whole debugging process. Below shown is our debugger option from the toolbar. Select the **Debug** tab, hover over **Windows** and select **Breakpoints**.



The other way to start the debugger at a specific location is to use the left side ribbon – a “Grey Band” in our SQL Server Management Studio script window. For simplicity reasons, I have shown this in the figure below. So as you can see, enabling the debugger is really easy now.

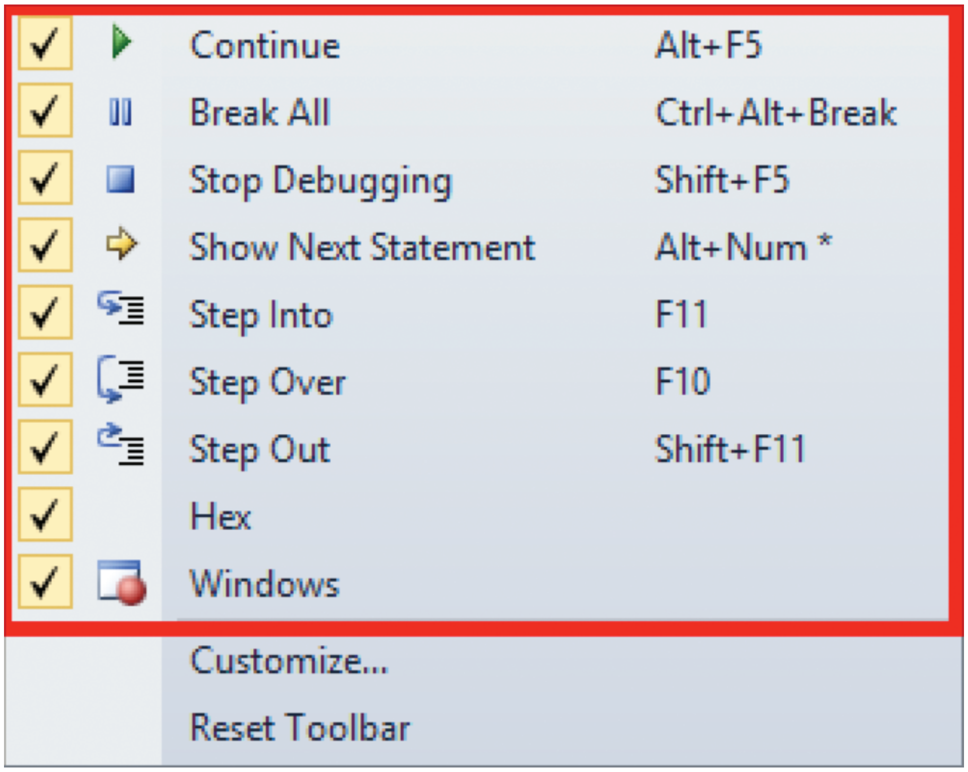


```
IF(@loop%200 = 0)
-- Used to track our values
PRINT @loop

SET @loop += 20
```

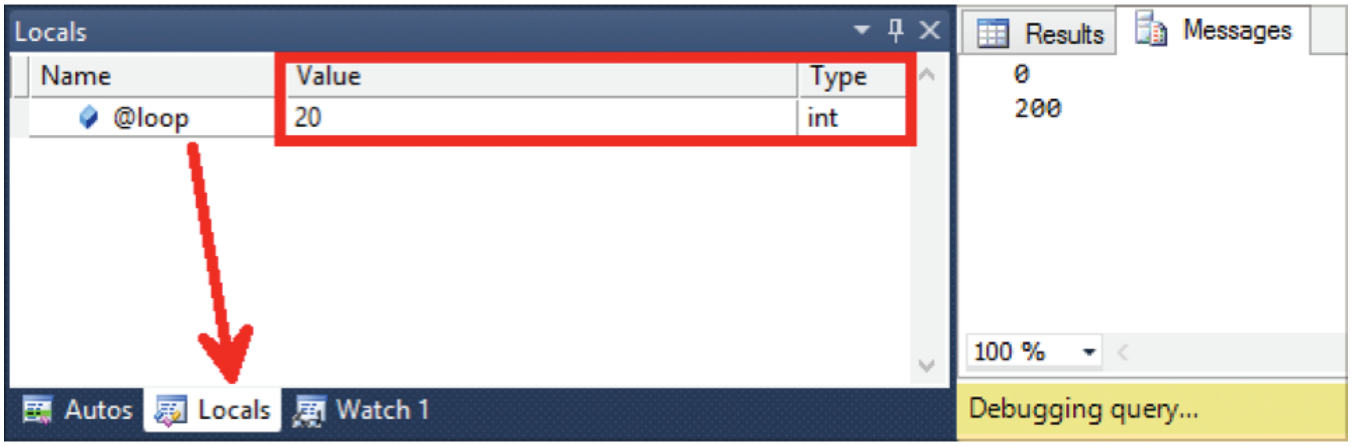
At Debugger.sql, line 9 character 5

Once the debugger has been enabled and we need to go through the code, we need to know what the other options available for debugging are. The toolbar shows us the list of options we can start using with the debugger. Some of them include **Break All**, **Stop Debugging**, **Next Statement**, **Step Into**, **Step Over** and so on.



WATCHING FOR VALUES

We know the values can be easily found using the *PRINT* statement inside our code block. But once the debugger is started, enable the Locals window. This is the window that allows us to look at all the local variables in the current batch. For our example, the local variable in the batch is @Loop.



Name	Value	Type
@loop	20	int

```
0
200
```

Debugging query...

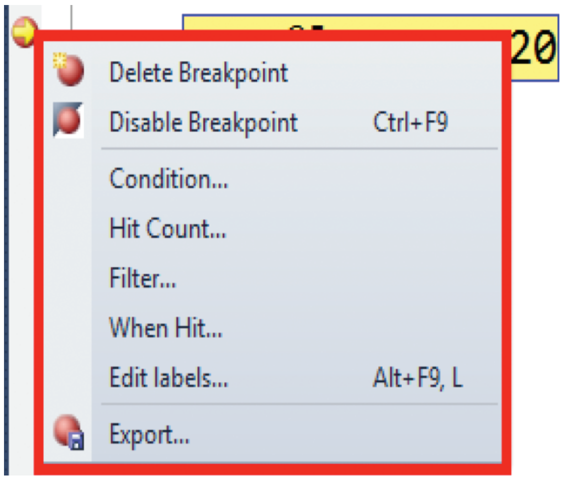
The same value as per the flow of code gets printed in the message window too as shown in the diagram above.

ADVANCED DEBUGGING OPTIONS

Watching for the locals is just one part of the story. As developers there are advanced options that we can start using with SQL Server Management Studio too. If we get an opportunity to right click the Debug point, we will be presented with a list of options that I would like to explain:

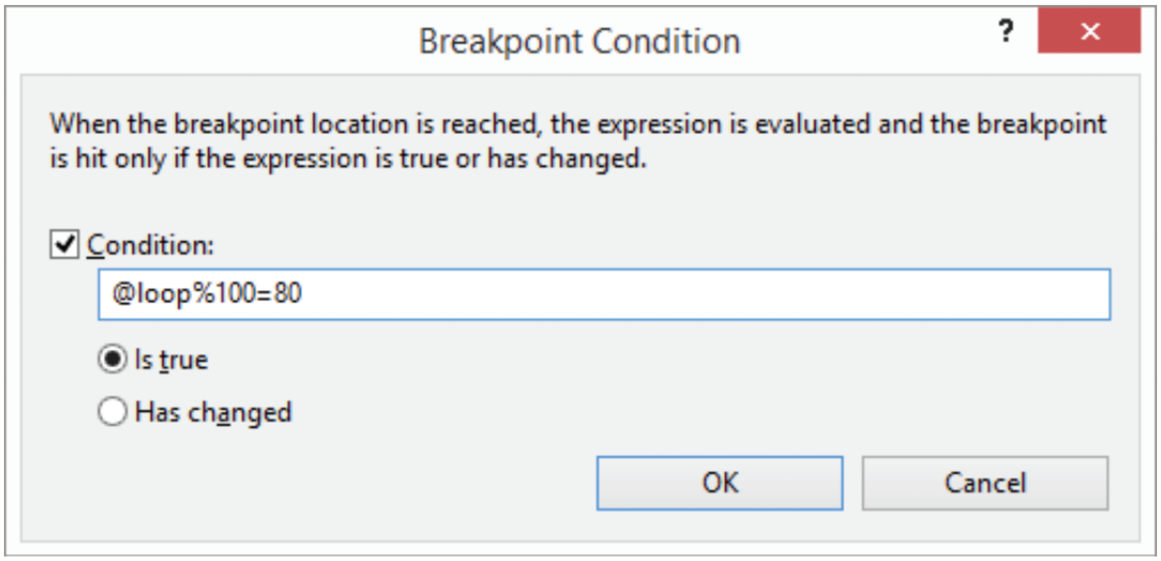
- Condition
- Hit Counter
- Filter
- When Hit

All of these options are handy and can be of great help to developers who are debugging their code. In the next section, we will talk about them quickly to understand how they can be used.



BREAKPOINT CONDITION

As soon as we select this option, we will get a dialog like the one shown below. Here we can add an additional condition which when satisfied the debugger will prompt. In our example we have put an additional condition that the debugger needs to stop when we hit multiples of 80.



When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

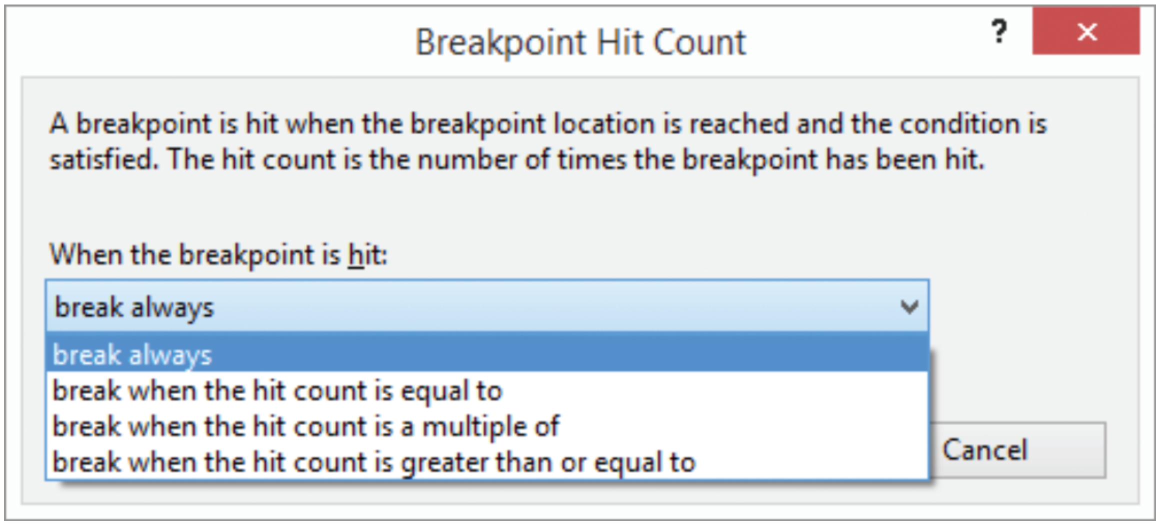
☑ Condition:

```
@loop%100=80
```

[✓ Is true] Has changed

BREAKPOINT HIT COUNT

This option lets us configure what needs to be done when the breakpoint is hit or when the codeflow reaches the breakpoint location. We have a number of options from “break always”, “when the hit count is equal to something”, “when count is a multiple of something” and so on. Use these if you want to skip to specific code flow when the variable hits one of these conditions.



Breakpoint Hit Count

A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.

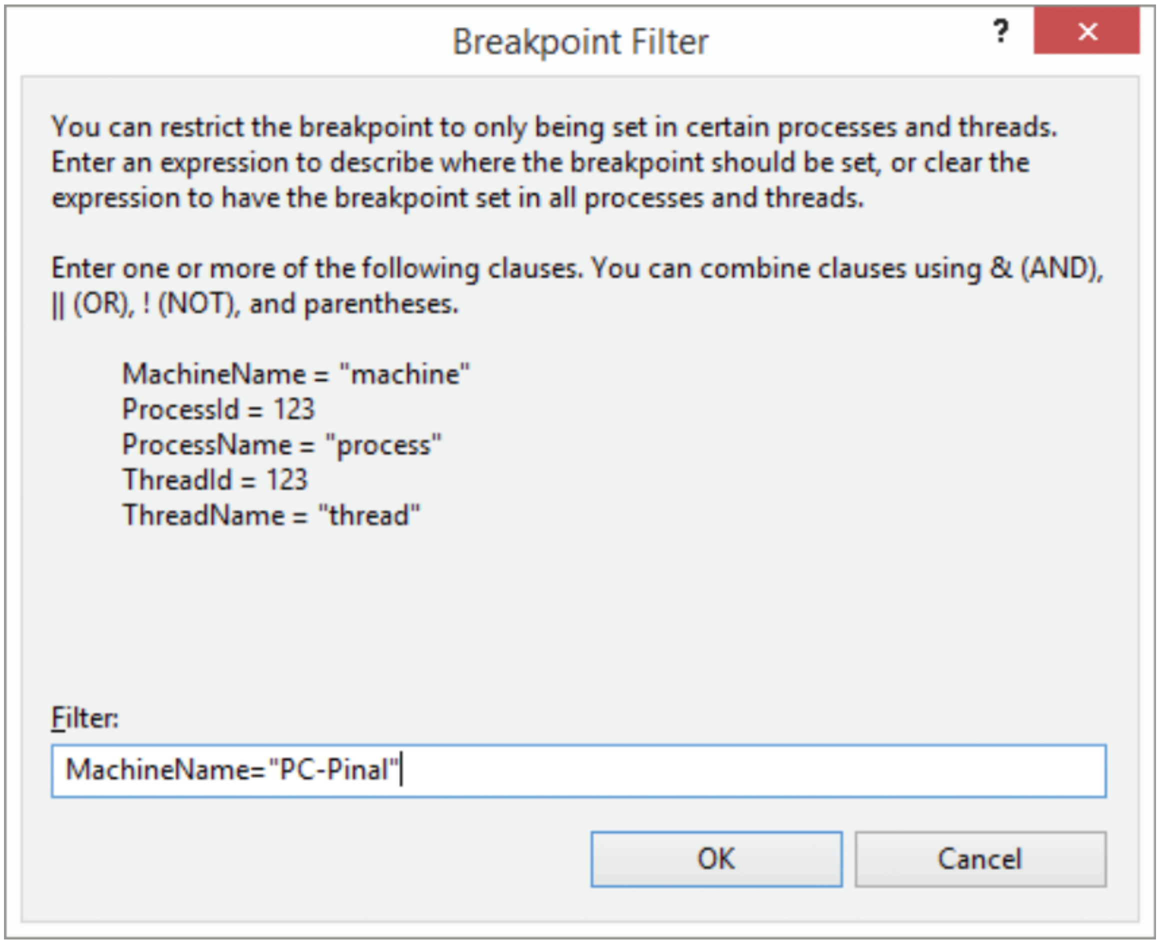
When the breakpoint is hit:

break always

- break always
- break when the hit count is equal to
- break when the hit count is a multiple of
- break when the hit count is greater than or equal to

BREAKPOINT FILTER

In this option for breakpoint, we are adding additional filters to the code block. Assume a case when the code is being run in a multi-user scenario. Here we would like to restrict the condition so that the code needs to be debugged only when the machine name is from Pinal. These additional filter blocks can be configured in this "Filter" block. In the below diagram we have added a filter to debug only when the machine is "PC-Pinal".



Breakpoint Filter

You can restrict the breakpoint to only being set in certain processes and threads. Enter an expression to describe where the breakpoint should be set, or clear the expression to have the breakpoint set in all processes and threads.

Enter one or more of the following clauses. You can combine clauses using & (AND), || (OR), ! (NOT), and parentheses.

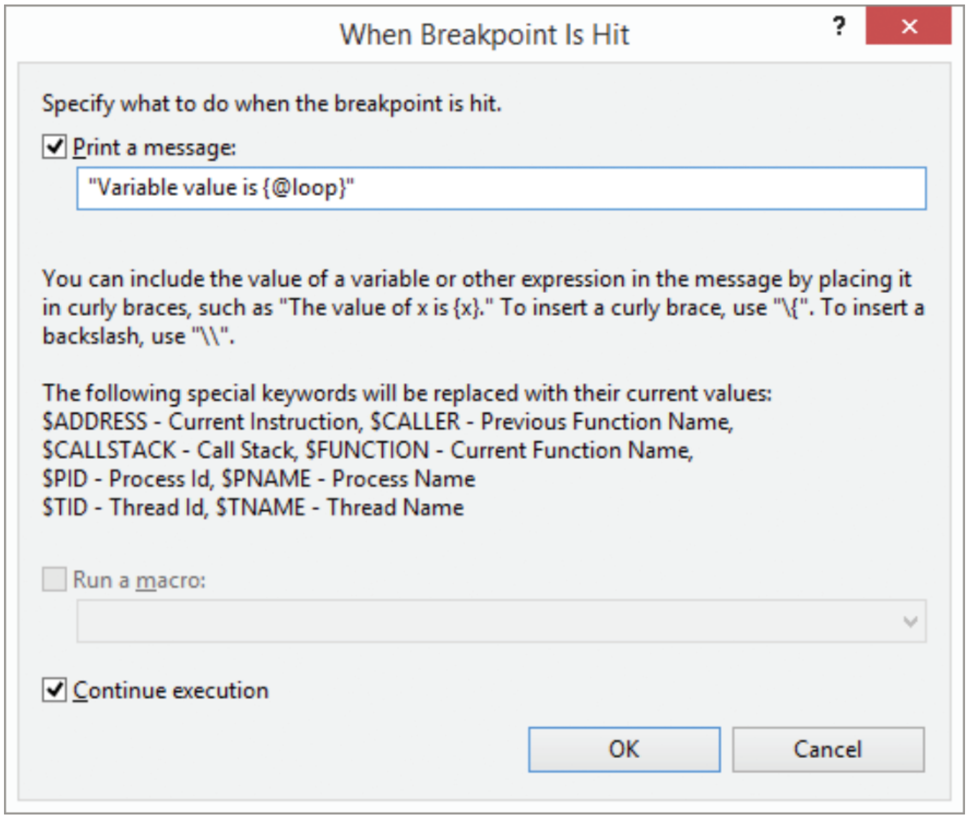
```
MachineName = "machine"
ProcessId = 123
ProcessName = "process"
ThreadId = 123
ThreadName = "thread"
```

Filter:

```
MachineName="PC-Pinal"
```

WHEN A BREAKPOINT IS HIT

This is like our very own conventional block of *PRINT* statement that we can also write in the options. It shows what can be done as we hit the breakpoint and how we can visualize the values differently. There are additional functions that can be added as part of our print which can vary from Process ID, Thread ID executing and so on. In our figure below, we have added the value of @Loop to be printed whenever we reach the breakpoint.



Specify what to do when the breakpoint is hit.

☑ Print a message:

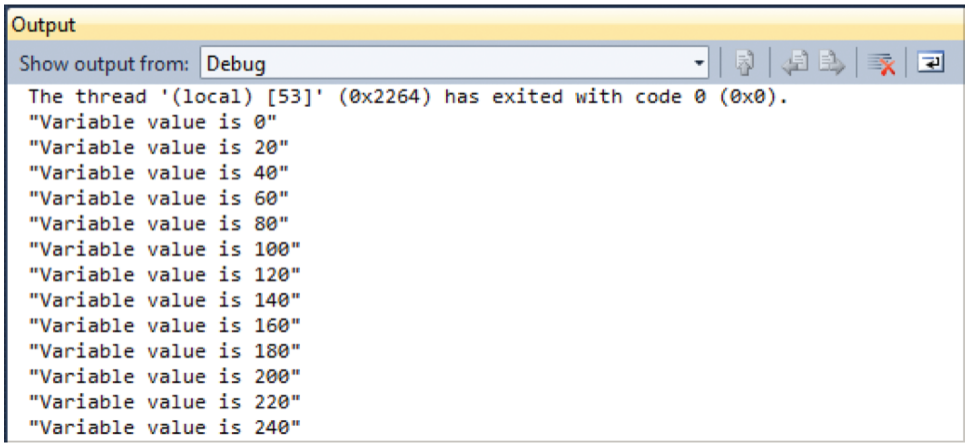
"Variable value is {@loop}"

You can include the value of a variable or other expression in the message by placing it in curly braces, such as "The value of x is {x}." To insert a curly brace, use "\{". To insert a backslash, use "\\".

The following special keywords will be replaced with their current values:
$ADDRESS - Current Instruction, $CALLER - Previous Function Name,
$CALLSTACK - Call Stack, $FUNCTION - Current Function Name,
$PID - Process Id, $PNAME - Process Name
$TID - Thread Id, $TNAME - Thread Name

☑ Continue execution

A typical output of the breakpoint is hit is shown below. We are printing all the values as we cross the Breakpoint location.



Show output from: Debug

```
The thread '(local) [53]' (0x2264) has exited with code 0 (0x0).
"Variable value is 0"
"Variable value is 20"
"Variable value is 40"
"Variable value is 60"
"Variable value is 80"
"Variable value is 100"
"Variable value is 120"
"Variable value is 140"
"Variable value is 160"
"Variable value is 180"
"Variable value is 200"
"Variable value is 220"
"Variable value is 240"
```

If we keep playing around with what is available with the Visual Studio projects, we can explore more options. As a developer of SQL Server, these are wonderful additions to the existing *PRINT* statements.

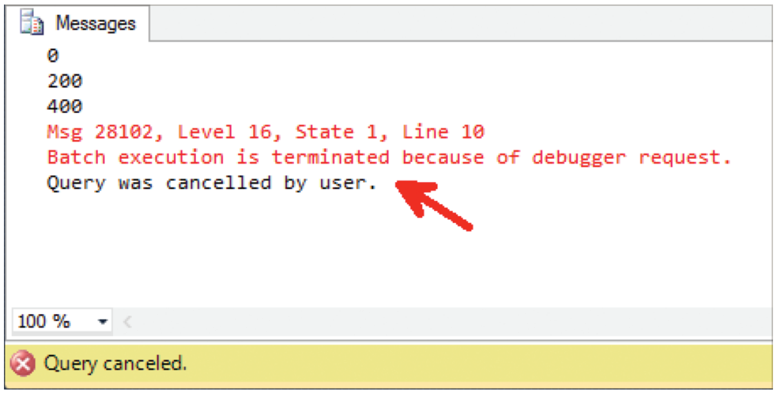
USEFUL DEBUGGER SHORTCUTS

We have shown a number of shortcut keys at various points in this whitepaper. Here is a summary of the shortcut keys.

Action	Shortcut
Toggle breakpoint	**F9**
Enable breakpoint	**CTRL+F9**
Stop debugging	**SHIFT+F5**
Step into	**F11**
Step over	**F10**
Step out	**SHIFT+F11**
Delete all breakpoints	**CTRL+SHIFT+F9**
Display the Breakpoints window	**CTRL+ALT+B**
Display the Locals window	**CTRL+ALT+V, L**
Display the Immediate window	**CTRL+ALT+I**
Display the Call Stack window	**CTRL+ALT+C**
Display the Threads window	**CTRL+ALT+H**
Start or continue debugging	**ALT+F5**
Show next statement	**ALT+NUM**

As we wrap up this concept, I would like to advise that the person needs to have *SYSADMIN* privileges to start debugging inside SQL Server. This is one of the pre-requisites to play with debugging capability. If the end user aborts the debugging session in the middle, then we will be presented with the below error message:

```
Msg 28102, Level 16, State 1, Line 10
Batch execution is terminated because of debugger request.
```



```
0
200
400
Msg 28102, Level 16, State 1, Line 10
Batch execution is terminated because of debugger request.
Query was cancelled by user.
```

CONCLUSION

Debugging, though a tough option inside SQL Server, has surely come a long way with the integration with Visual Studio to make SQL Server Management Studio much more powerful. These additions to SSMS have surely increased the productivity of Developers and DBAs alike.

SECTION FOUR
NUTS AND BOLTS OF PERMISSIONS AND SECURITY

INTRODUCTION

Being in the computer industry is one of the most challenging things anyone can get into. When upgrading one's knowledge to keep the data secure, there are a number of hurdles to cross. In this industry where we look for quick, fast responses from development to deployment to sales, everyone is expected to deliver without compromising any of the business parameters. As we chase around the tough deadlines for delivery, the team tries to take shortcuts. This, for most cases, involves cutting corners and compromising on designing for secure systems. This is a concern for a number of organizations. DBAs love to build a secure system to deploy applications delivered from the Application team. Having said that, in this whitepaper I would like to get back to the basics of security. We want to cover some of the fundamental building blocks that were added with SQL Server since the 2005 version that are worth noting.

PRINCIPALS

From SQL Server 2005 we refer to all login and database user accounts as "Principals". This is consistent with the Windows concept of a "security principal". A security principal is an entity that can be identified and verified via authentication. The concept hasn't changed much from earlier versions of SQL Server; it is more of a terminology change.

The capabilities of principals depend on their scope of definition (server or database) and whether they represent a single principal (Primary) or a collection of principals (Secondary). Each Principal is uniquely identified by a Security Identifier (SID).

SERVER-LEVEL PRIMARY PRINCIPALS

DMVs are the heart of finding what type of login is currently being used. At a high-level, to learn about the Primary Principals, we can use the following DMV query.

```sql
SELECT * FROM master.sys.server_principals;
```

Windows Login

Windows logins are defined in the local machine or the AD forest (called a domain login) of which the SQL server instance is a part of. The management of these principals is not within the scope of SQL Server. Each Windows login is uniquely represented with a SID, provided by Windows itself. These can be obtained using the following query:

```sql
SELECT * FROM master.sys.server_principals WHERE type = 'U'
```

SQL Login

SQL logins are defined in a particular SQL Server instance. They are undefined outside the scope of that SQL Server. They are used for authorization of resources within that server. However, they are visible across all databases in the server and can be used for authorization of resources in any database in the server. As with Windows logins, each SQL login is uniquely represented by a SID (which in this case is a GUID as it has no Windows-provided SID).

```
SELECT * FROM master.sys.server_principals WHERE type = 'S'
```

SERVER-LEVEL SECONDARY PRINCIPALS

As we discuss the fine print, these are two kinds of Server level secondary principals as described below.

Windows Groups

Windows groups are again defined in the local machine or the AD forest (called domain login) of which the SQL Server instance is a part. The management of these principals is not within the scope of SQL Server. Based on the nature of the domain there can be different types of these. Each Windows group is uniquely represented by a SID. We can query them using:

```
SELECT * FROM master.sys.server_principals WHERE type = 'G'
```

SQL Role

SQL roles are specifically more widely known as “fixed server roles”. They are defined in each SQL Server instance and membership is undefined outside the scope of that SQL Server. They are used for authorization of resources within that server. As with Windows logins, each SQL role is uniquely represented by a SID (which in this case is a GUID as it has no Windows-provided SID).

```
SELECT * FROM master.sys.server_principals WHERE type = 'R'
```

DATABASE-LEVEL PRIMARY PRINCIPALS

Database User

Any server-level principal which requires access to a database needs to be mapped to a Database User in the relevant database. A Database User is a principal at the database level. Every database user is a member of the public role. SQL Server includes a guest database user account that has limited access rights. With the secure by default strategy, the latest versions of SQL Server keep this account in a disabled status.

```
SELECT * FROM <dbname>.sys.database_principals WHERE type in ('S','U','C')
```

Application Role

Application roles are database scoped principals that are like users, but have no mapped login. They are restricted to a database and are identified by a password.

```
SELECT * FROM <dbname>.sys.database_principals WHERE type = 'A'
```

We also have something called a *CONTAINED DATABASE USER* which is worth a mention here. We are keeping it out of scope of this document because it is all by itself a larger topic to discuss.

DATABASE-LEVEL SECONDARY PRINCIPALS

Database Role

Database roles are an aggregation mechanism for collections of database users. Since they are defined within the database, they cease to exist outside the database. Members of the *DB_OWNER* and *DB_SECURITYADMIN* fixed database roles can manage fixed database role membership. Every database user is a member of the public database role. When a user hasn't been granted or denied permissions on a securable, it inherits the permissions granted to the public on that securable.

```
SELECT * FROM <dbname>.sys.database_principals WHERE type = 'R'
```

LOGINS AND USERS

As we have discussed, the concept of a "login" as an object has been replaced by the "principal" object. However, we still have to be able to provide the ability for SQL Server to recognize a principal and authorize it for login access to SQL Server and its databases. To do this we create a login in SQL Server for the relevant principal and we create a user in each required database and map it to the principal.

When we create a login, we are adding an entry into the *SYS.SERVER_PRINCIPALS* view in master in much the same way as used to occur with the sysxlogins system table in earlier versions of SQL. There is a new syntax, *CREATE LOGIN* to replace the previous *SP_ADDLOGIN* procedure.

When we create a database user, we are adding an entry into the *SYS.DATABASE_PRINCIPALS* view in a specific database, in much the same way as used to occur with the sysusers table in earlier versions of SQL Server. One of these views exists in each database, as database users are scoped at database level. There is a new syntax, *CREATE USER*, to replace the previous *SP_ADDUSER* procedure.

It is important to be aware of a number of new features though, such as the ability to map a SQL Login to a Certificate or Asymmetric key. Additionally it is also possible to assign credentials to a SQL login, which is used for the mapping of a SQL login to external credential such as a Windows login for authentication and authorization purposes when accessing external resources. It is also now possible to *ENABLE* and *DISABLE* a login.

PASSWORD POLICY

There are also a number of new options for SQL logins which are worth pointing out, *CHECK_EXPIRATION* and *CHECK_POLICY* being two of those. These specify whether password expiration policy should be enforced on this login and also whether the Windows password policies of the computer on which SQL Server is running should be enforced on this login.

In order to strengthen the security of SQL Login accounts, since SQL Server 2005 we have the ability to enforce strong passwords, account lockout, password age mechanisms and/or enforcement of domain or local level password policy.

The Group Policy security settings can be examined by running *GPEDIT.MSC*, and expanding **Computer Configuration** → **Windows Settings** → **Security Settings** → **Account Policies** → **Password Policy**. If I do this on my machine I see the following settings (note your values may have different settings):

Enforce password history	8 passwords remembered
Maximum password age	90 days
Minimum password age	1 days
Minimum password length	8 characters
Password must meet complexity requirements	Enabled
Store password using reversible encryption for all users in the domain	Enabled

Local security policy for a non-domain computer can be examined using the **Local Security Settings** tool under **Administrative Tools**.

It is these settings that Password Policy is enforcing for SQL logins. Of course Windows logins already have this enforced by the system, so in effect we are only bringing SQL in line with established best practice. Note that Password Policy does not apply to Application Roles in this release. *CHECK_POLICY* is enabled by default, although *CHECK_EXPIRATION* is not.

CHECK_EXPIRATION enforces minimum and maximum password age.
CHECK_POLICY enforces all other policies.

When you run afoul of any of the policy settings, your account is locked. It must be unlocked by a Sysadmin, using the *UNLOCK* option of *ALTER LOGIN*.

PASSWORD POLICY TROUBLESHOOTING

Most of the error messages that cause login failures due to Password Policy enforcement are fairly clear, however there may be circumstances under which the error is not correctly propagated to the user, due to legacy applications, poor error handling, etc. In these cases the EventLog and Errorlog should contain the relevant message.

SQL logins may expire after the Maximum Password age. In order to address this and identify accounts which are affected by the policy, a Sysadmin can run the following script to provide the current state of SQL logins:

```
DECLARE @max_age int
SET @max_age = 90

SELECT name,
      is_policy_checked,
      is_expiration_checked,
        LOGINPROPERTY(name, 'IsExpired') AS is_expired,
        LOGINPROPERTY(name, 'IsLocked') AS is_locked,
        LOGINPROPERTY(name, 'IsMustChange') AS is_mustchange,
        LOGINPROPERTY(name, 'HistoryLength') AS history_length,
        LOGINPROPERTY(name, 'BadPasswordCount') AS bad_password_count,
        DATEDIFF(day, GETDATE(), DATEADD(day, @max_age,
                   CAST(LOGINPROPERTY(name, 'PasswordLastSetTime')
                   AS datetime))) AS days_until_expiration,
        LOGINPROPERTY(name, 'PasswordLastSetTime') AS
                   password_last_set_time,
        LOGINPROPERTY(name, 'BadPasswordTime') AS bad_password_time,
        LOGINPROPERTY(name, 'LockoutTime') AS lockout_time
FROM sys.sql_logins
```

Note that we are required to set the *MAX_AGE* variable manually. This should be set to the value for maximum password age as per the Global Policy/Local Policy currently in force on the SQL Server machine, which can be obtained via *GPEDIT.MSC* or Local Security settings.

In order to unlock an account, a Sysadmin should run the following statement to unlock the affected user account:

```
ALTER LOGIN Pinal WITH PASSWORD = 'password' UNLOCK
GO
```

SPECIAL USER *SYSADMIN*

If SQL is installed in Windows Authentication mode we assign a random password to the *SA* account and disable it by default. This is to prevent the situation where we have a system with an active *SA* account with a blank password. If the authentication mode is changed, the *SA* account needs to be re-enabled and a password must be assigned to it.

```
ALTER LOGIN sa ENABLE;
ALTER LOGIN sa WITH PASSWORD = 'myC0mplexPa$$w0rd'
```

In order to do this you must be a member of the *SYSADMIN* role. If you have dropped the BUILTIN\Administrators group from the *SYSADMIN* role and have no other Sysadmins other than *SA* then there is a clear problem. In order to overcome this, it should be possible to connect via DAC as a member of the server group BUILTIN\Administrators (using Windows Authentication) and add a *SYSADMIN* account or to re-enable the *SA* account and reset its password. Any other action that this Admin attempts inside SQL Server will work only if that Admin account has been granted access to perform that action.

CONCLUSION

Security is a core area and non-negotiable when it comes to mission critical applications. As DBAs, knowing the fine print of working with security is very important. In this whitepaper, we got a chance to explore and learn some of these basics so that our foundation is strong. I highly recommend we keep our eyes open on our deployment server to confirm our environments are configured correctly. Look for possible security loopholes and try to plug them before they become a compliance issue.

SECTION FIVE
SQL SERVER SECURITY PRACTICES

INTRODUCTION

There are a number of examples where data theft has brought businesses to a halt or resulted in bad press that will leave a tarnished image. For example, in 2013 Target announced about 40 million credit and debit card details were stolen. In a similar timeframe, Adobe claimed that the corporate database of 130 million user records was stolen. There are many more horror stories from banks, financial institutions, and retail organizations in the past and they get even scarier as we move into this digital only world. As organizations start looking at security as a first class citizen and work on it proactively, these incidents will still keep happening and we will always be playing the catch-up game.

If you are in the job of consulting or if you are in sales oriented jobs, the likelihood of engaging with new customers on a daily basis is inevitable – and I am one of them. This is something I do a lot in my day job and sometimes I have to deal with some really rude security procedures at the clients' place of business. I always wonder why we get grilled, frisked and put under so much scrutiny. In one of my recent trips while hopping via airports, the security guard stopped me and told me to write my phone number and name on my baggage tag. I first thought about it and then out of curiosity asked him politely (in frustration), "Why are you doing this and forcing me? I am a frequent traveler and have never been asked to do this in my 8+ years of travel anywhere. This is just a domestic trip, and please relax, because I have a flight to catch."

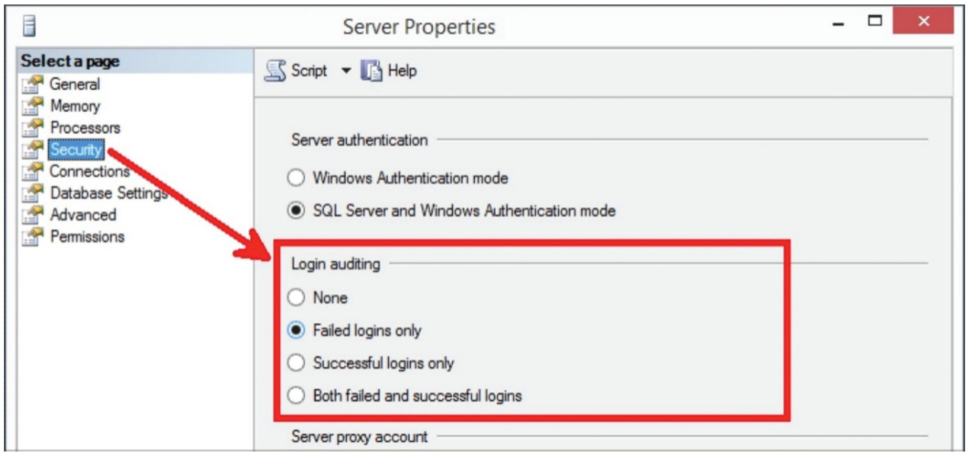
The guard smiled and politely said, "All the more reason, sir – you now need to write it every time you plan to travel." Seeing me getting a little impatient, he added, "Sir, since you are in a hurry and you are distributing your laptop, bag, cellphone, etc. in different bins, it is quite possible you might miss picking up your materials because of the rush. So if you had written down the phone number, we can reach you individually rather than making a public announcement which can be of inconvenience to all in this airport." He added to say, "Sir, I am just doing my job."

This incident changed the way I see security personnel. They are doing a job and we need to just respect them. A deep introspection got me into thinking about security in a different way. What about security in the software we develop? Why is security not a consideration during the design phase itself? Securing our data is one of the most important aspects when it comes to keeping your trade secrets from preying competition.

In this paper let me take some of the common security practices that I propose to my customers on a daily basis and how one needs to start implementing security measures within their deployment of SQL Server. I am sure if you are a DBA, these are like checklists you don't want to miss when working with a database engine like SQL Server. We will cover details around the logins and authentication area.

AUDIT LOGINS

Keep the setting of "Both failed and successful logins" for login auditing. It is critical we track or at least keep an eye on who is accessing the system or attempting to access the system. By default, these settings are not to be tampered with. But I have seen sometimes inexperienced DBAs do turn this setting off. The default is to Audit the "Failed logins only", but in mission critical systems it might be a better practice to audit both failed and successful logins.



Server authentication

○ Windows Authentication mode

● SQL Server and Windows Authentication mode

Login auditing

○ None

● Failed logins only

○ Successful logins only

○ Both failed and successful logins

Server proxy account

The idea is to make sure there are no DOS attacks made on the system and we are tracking the connections coming into the server. From SQL Server 2008, it is also possible to use the new Audit feature to log connection attempts. If you plan to use this, as a practice use a secured location for audit data files.

AUTHENTICATION AND DISABLE *SA*

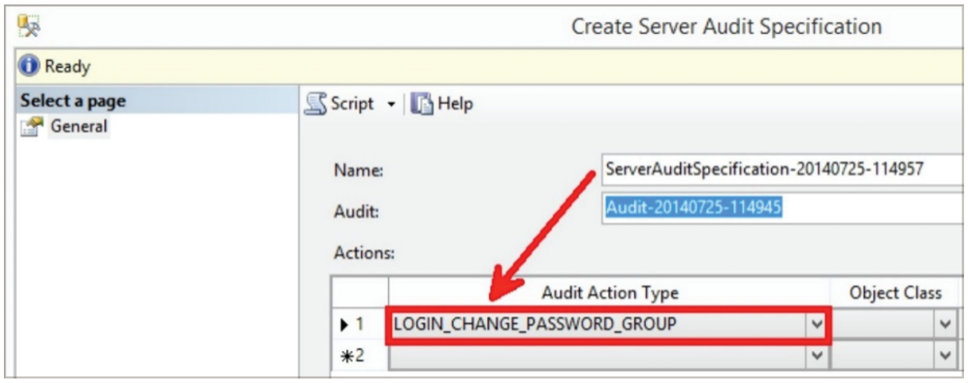
Try to use Windows Authentication as much as possible. Usage of mixed mode authentication is recommended for legacy applications and non-Windows users. I recommend this religiously because when multiple users use the same login, it becomes difficult to know which “NT User” actually did an operation on the server. I recommend this for SQL Server Administration too, instead of using the default *SYSADMIN SA* account. Try not to share the *SA* account across multiple users. We can change the authentication type on the **Server Properties → Security → Server Authentication**; this setting can be found as shown in above figure.

If possible make sure to disable the *SA* login inside SQL Server. This is because the user account can be easily guessed by anyone working with SQL Server. Ideally, rename or disable this account and use some other account to administer your SQL Server instance. The following command can help you *DISABLE / ENABLE* the *SA* user account.

```
/* Disable SA Login */
ALTER LOGIN [sa] DISABLE
GO
/* Enable SA Login */
ALTER LOGIN [sa] ENABLE
GO
```

As we move to the next recommendation, if you have to use mixed mode authentication then I highly recommend enabling the password expiration and password complexity values for all the logins created on the server for all the authentication types.

If there are multiple users administering a given SQL Server instance, then it is recommended to track password changes for SQL Server logins. Enabling auditing on password changes for SQL Server logins allows you to investigate when and by whom the passwords of specific logins have changed. Consider creating a server audit for the action '*LOGIN_CHANGE_PASS-WORD_GROUP*'.



Name: ServerAuditSpecification-20140725-114957
Audit: Audit-20140725-114945
Actions:

	Audit Action Type	Object Class
1	LOGIN_CHANGE_PASSWORD_GROUP	
*2		

VIEW DATABASE PERMISSION

This is a server level permission and lesser appreciated permission. If this permission is given then the login can see all the metadata for all databases irrespective of the login owning the database or not. By controlling this permission we effectively restrict the amount of data a login can see in the *SYS.DATABASES*, *SYS.SYSDATABASES* views and *SP_HELPDB* calls. To limit visibility to database metadata, deny a login the *VIEW ANY DATABASE* permission.

```
/* REVOKE VIEW DEFINITION PERMISSION */
USE master
GO
REVOKE VIEW ANY DEFINITION TO Pinal
```

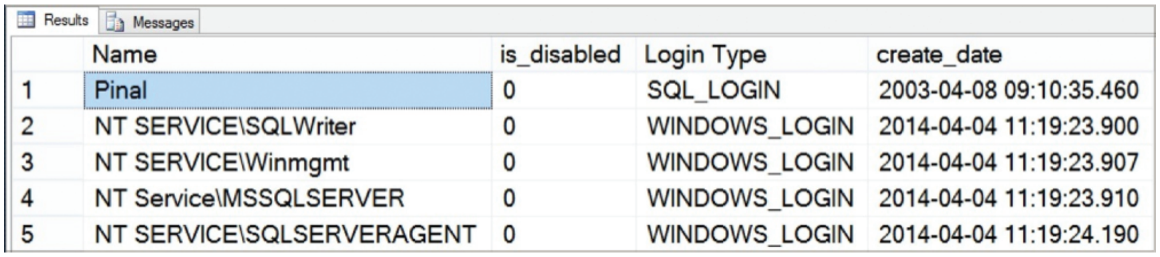
After this permission is denied, a login can see only metadata for master, tempdb and databases that the login owns. By default the public role is granted this permission, don't use *DENY* on the "public" role otherwise no one will be able to list database objects.

SYSADMINS HAVE SUPER POWERS

From SQL Server 2008 onwards, the Windows built-in administrators group is not part of the SQL Server *SYSADMIN* fixed server role. This is a great "secure by default" strategy and best practice in my opinion and I am glad this was done. Now during the installation, we are requested to explicitly mention / grant the logins who will have *SYSADMIN* privileges. Be very cautious in giving *SYSADMIN* privileges to logins. The below script lets you find out who are the Sysadmins in your server today.

```
SELECT
prin2.name AS [Name],
prin2.is_disabled,
prin2.type_desc AS [Login Type],
prin2.create_date
FROM sys.server_principals prin
INNER JOIN sys.server_role_members mem
ON prin.principal_id = mem.role_principal_id
INNER JOIN sys.server_principals prin2
ON prin2.principal_id = mem.member_principal_id
WHERE prin.type = 'R' and prin.name = N'sysadmin'
```

Sysadmins as the name suggests have all the permissions to do anything inside SQL Server. If you have to give permission, then provision the logins in the *SYSADMIN* fixed server role directly; don't use a common SQL Authentication with the *SYSADMIN* rights on the server – this helps while auditing. On my local server you can see I have hardly given rights to anyone apart from my own Login.



	Name	is_disabled	Login Type	create_date
1	Pinal	0	SQL_LOGIN	2003-04-08 09:10:35.460
2	NT SERVICE\SQLWriter	0	WINDOWS_LOGIN	2014-04-04 11:19:23.900
3	NT SERVICE\Winmgmt	0	WINDOWS_LOGIN	2014-04-04 11:19:23.907
4	NT Service\MSSQLSERVER	0	WINDOWS_LOGIN	2014-04-04 11:19:23.910
5	NT SERVICE\SQLSERVERAGENT	0	WINDOWS_LOGIN	2014-04-04 11:19:24.190

It's highly recommended to give *SYSADMIN* privileges to the lowest number of accounts to maximize security. And do not assign any elevated rights to a user if that user does not need it.

HOW TO PROTECT FROM SYSADMINS?

As we can see, the Sysadmins are really powerful, and when using *SYSADMIN* privileges, note that no permission check is done. So use this with caution and give access to only limited and, if possible, the least number of users.

Organizations have a need to protect their data and I have seen Management asking if there is a way to protect data even from DBAs. The first question I ask is, "What rights have you given them?" The standard answer I get is, "They are DBAs and hence we have given them the administrative rights of Sysadmins." If you look at the examples and analogy I gave before, this is surely dangerous, right? So the immediate question is, what should be the rights for a DBA?

These are tough questions but with every release of SQL Server there have been enhancements that we don't want to miss. Let me build the set of permissions for you leading to a feature introduced in SQL Server 2014.

Let us first start creating our super user SQLAuthority.

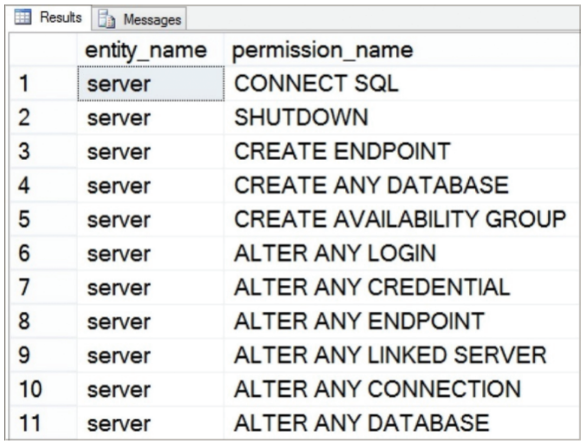
```
USE [master]
GO
CREATE LOGIN SQLAuthority WITH PASSWORD=N'pass@word1',
DEFAULT_DATABASE=[master],
CHECK_EXPIRATION=ON, CHECK_POLICY=ON
GO
```

The idea now is NOT to give *SYSADMIN* rights but to give rights that still enable the DBA to do their day-to-day activities without any problem. We are going to give this user *CONTROL SERVER* rights.

```
GRANT CONTROL SERVER TO SQLAuthority;
GO
```

Let us query the permissions DMV to find out the effective permission SQLAuthority has after this *GRANT*. We will be using the following command:

```
SELECT entity_name, permission_name
FROM sys.fn_my_permissions(NULL, NULL);
```



Results | Messages

	entity_name	permission_name
1	server	CONNECT SQL
2	server	SHUTDOWN
3	server	CREATE ENDPOINT
4	server	CREATE ANY DATABASE
5	server	CREATE AVAILABILITY GROUP
6	server	ALTER ANY LOGIN
7	server	ALTER ANY CREDENTIAL
8	server	ALTER ANY ENDPOINT
9	server	ALTER ANY LINKED SERVER
10	server	ALTER ANY CONNECTION
11	server	ALTER ANY DATABASE

This gives us explicitly 34 specific server level permissions. This still isn't that restrictive. If SQLAuthority tries to elevate himself to *SYSADMIN* rights using the following command:

```
ALTER SERVER ROLE sysadmin
ADD MEMBER SQLAuthority
```

He will be presented with an error stating no permissions.

```
Msg 15151, Level 16, State 1, Line 11
Cannot alter the server role 'sysadmin',
because it does not exist or you do not have permission.
```

But from the list of permissions granted, we can find that *IMPERSONATE ANY LOGIN* is available for this user. If that is the case, then SQLAuthority can impersonate ANY user in the system and gain access to the server. Typically, he would do it this way:

```
EXECUTE AS LOGIN = 'sa';

ALTER SERVER ROLE sysadmin
ADD MEMBER SQLAuthority

REVERT;
```

That is it, now SQLAuthority has gained *SYSADMIN* rights into the system, and this seems to be a problem prior to SQL Server 2012. With SQL Server 2012, there was an interesting addition to *DENY* impersonation rights to a user, and even if they have CONTROL SERVER rights, SQL Server will honour the *DENY* permission. In our example, we can explicitly deny Impersonation to SQLAuthority:

```
DENY IMPERSONATE ON LOGIN::sa TO [SQLAuthority]
```

This is great, and now SQLAuthority cannot impersonate the *SA* account. But do you see the problem here? We have not solved the problem yet. We have explicitly denied impersonation to this account, but there can be other users inside the system who have *SYSADMIN* rights and our *CONTROL SERVER* rights will give SQLAuthority the opportunity to impersonate as one of them. It is humanly impossible to *DENY* such rights to each and every user with *SYSADMIN* rights for SQLAuthority.

SQL Server 2014 comes with a permission called *IMPERSONATE ANY LOGIN* rights. This is powerful for the scenario we questioned. After giving the *CONTROL SERVER* rights, we can issue the following command in SQL Server 2014:

```
DENY IMPERSONATE ANY LOGIN TO SQLAuthority
GO
```

After this command the effective permission for SQLAuthority is the same as *CONTROL SERVER* but he has been explicitly denied *IMPERSONATION* rights. This is so powerful because as the management wants, we can give close to *SYSADMIN* rights to a user to perform his daily duties yet not give them the ability to elevate themselves easily.

If we run the command for effective permissions, we can see the list now has one fewer row at 33. The DMF for this is:

```
SELECT entity_name, permission_name
FROM sys.fn_my_permissions(NULL, NULL)
```

Also worth a mention, the other permissions added in SQL Server 2014 are *CONNECT ANY DATABASE* and *SELECT ALL USER SECURABLES*.

```
use [master]
GO
GRANT CONNECT ANY DATABASE TO [SQLAuthority]
GO
GRANT SELECT ALL USER SECURABLES TO [SQLAuthority]
GO
```

Think of these rights when you want to give permission to an Auditor who visits your organization for compliance and wants access to the Server while they can only select and not update anything. These rights are powerful for those scenarios.

CONCLUSION

Security is a core area and non-negotiable when it comes to mission critical applications. We know how companies have lost business and lost respect in the press because of lack of security measures. With every release of SQL Server, there are tons of additions that get added as part of the platform. It is important to start using them in our application deployment design so that these loopholes can be avoided. In this paper we discussed some of the general best practices as they evolved with SQL Server versions and we took a note of specific *SYSADMIN* privileges and how we can secure our server using some of the permissions added in SQL Server 2014.

ABOUT THE AUTHOR:

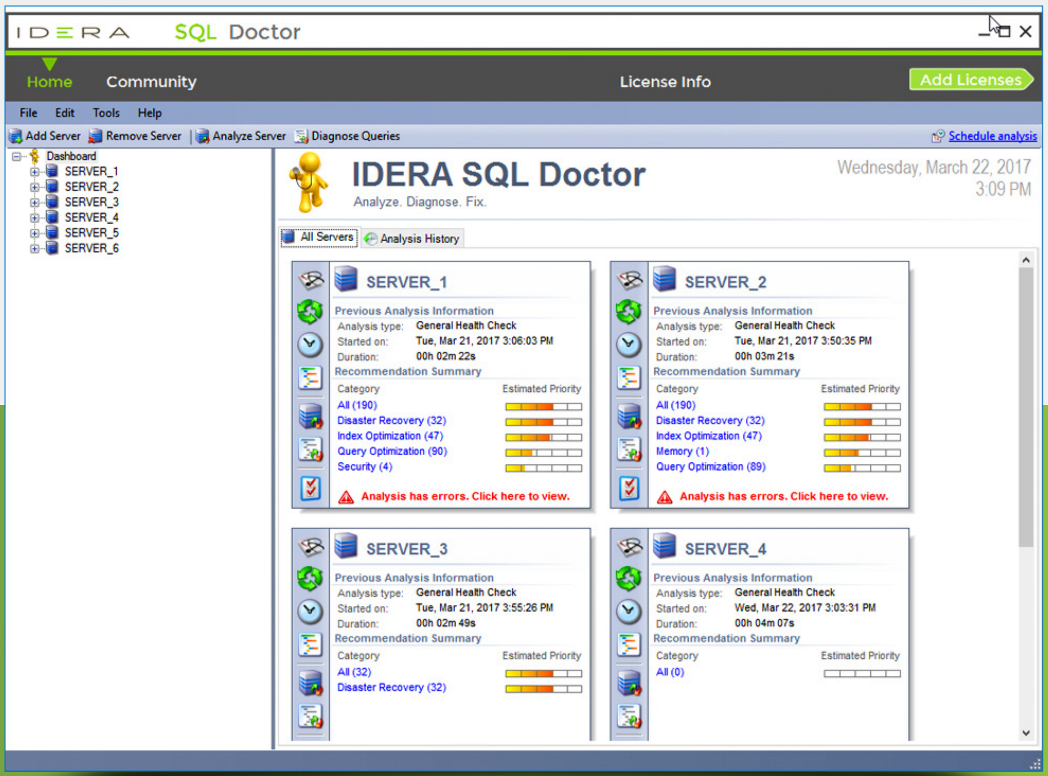
Pinal Dave is a Developer Evangelist. He has authored 11 SQL Server database books, 15 Pluralsight courses and has written over 3000 articles on the database technology on his blog at http://blog.sqlauthority.com. Along with 10+ years of hands on experience he holds a Masters of Science degree and a number of certifications, including MCTS, MCDBA and MCAD (.NET). His past work experiences include Technology Evangelist at Microsoft and Sr. Consultant at SolidQ.

SQL Doctor

TUNE DATABASE PERFORMANCE, DISASTER RECOVERY, AND SECURITY WITH EXPERT RECOMMENDATIONS

- Tune SQL Server for physical, virtual, and cloud environments
- View a summary of the health of the entire enterprise and a selected server
- Display prioritized list of performance optimization recommendations
- Generate executable SQL scripts to optimize performance
- Access analysis history and view trends
- View advice for SQL Server 2016 and 2017, Azure SQL Database, and Amazon RDS

Start for FREE



SERVER_1

Previous Analysis Information
Analysis type: General Health Check
Started on: Tue, Mar 21, 2017 3:06:03 PM
Duration: 00h 02m 22s

Recommendation Summary

Category	Estimated Priority
All (190)	
Disaster Recovery (32)	
Index Optimization (47)	
Query Optimization (90)	
Security (4)	

Analysis has errors. Click here to view.

SERVER_2

Previous Analysis Information
Analysis type: General Health Check
Started on: Tue, Mar 21, 2017 3:50:35 PM
Duration: 00h 03m 21s

Recommendation Summary

Category	Estimated Priority
All (190)	
Disaster Recovery (32)	
Index Optimization (47)	
Memory (1)	
Query Optimization (89)	

Analysis has errors. Click here to view.

SERVER_3

Previous Analysis Information
Analysis type: General Health Check
Started on: Tue, Mar 21, 2017 3:55:26 PM
Duration: 00h 02m 49s

Recommendation Summary

Category	Estimated Priority
All (32)	
Disaster Recovery (32)	

SERVER_4

Previous Analysis Information
Analysis type: General Health Check
Started on: Wed, Mar 22, 2017 3:03:31 PM
Duration: 00h 04m 07s

Recommendation Summary

Category	Estimated Priority
All (0)	

IDERA

IDERA.com