“The SQL Server and .NET Blog” eBook Series

Administering SQL Server

Artemakis Artemiou

“The SQL Server and .NET Blog” eBook Series

Administering SQL Server

ARTEMAKIS ARTEMIOU

PUBLISHED BY
Artemakis Artemiou

Copyright © 2013 Artemakis Artemiou

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at
http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the author, nor distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

About the Author

Artemakis Artemiou is a Senior SQL Server Architect and a SQL Server MVP. Artemakis started working with SQL Server more than ten years ago. He holds many certifications on different versions of SQL Server and he is the President of the Cyprus .NET User Group (CDNUG) and the INETA-Europe Country Leader for Cyprus. He regularly writes articles on different topics on SQL Server and publishes them on his blog or on other blogs/websites as guest articles. Artemakis participates as a speaker on many local user group events as well as an invited speaker on Microsoft conferences and workshops. Furthermore, he is the owner of "The SQL Server and .NET Blog" and "The SQL Server and .NET TV".

Some of the online channels where you can find Artemakis are:

- Twitter: http://twitter.com/artemakis
- The SQL Server and .NET Blog: http://aartemiou.blogspot.com/
- The SQL Server and .NET TV: http://www.youtube.com/sqlserverdotnetblog/
- Artemakis Artemiou's SQL Server Books: http://sqlbooks.aartemiou.com/
- Artemakis Artemiou's SQL Tools: http://sqltools.aartemiou.com/
- Personal website: http://www.aartemiou.com/
- CodePlex: http://www.codeplex.com/site/users/view/ArtSQL

Contact the Author

You can contact the author by email at: artemakis@aartemiou.com

To all of those who dare to experiment with new technology,
who constantly try to become better in what they do,
but never lose respect for their fellow human being in the process.
-A.A.

Table of Contents

Introduction

I started working with SQL Server and .NET (C#) more than ten years ago. Since then it has been quite a journey! Each release came - and still comes - with exciting new features enabling us to do more and more! Every time waiting for that new built, in order to start testing it, exploring it, learning it, as soon as it becomes available. The possibilities are endless! The only limit is your creativity!

The massive interaction with the SQL Server community started at about seven years ago. Blogging, organizing user group events, speaking in user group meetings, conferences and other events, open-source projects related to SQL Server, guest articles, discussions on message boards/forums, and much more!

The love for technical writing and knowledge sharing urged me for adding another activity to my interaction with the community that is ***book authoring***. In order to be able to write, you first need to acquire and comprehend the specific technical knowledge. You need to explore, to experiment, to test. You need to test the limits of each new technology or feature in order to be able to fully understand its nature and capabilities.

SQL Server is a powerful data platform that includes several data management and analysis technologies that allow you to do just about anything. Since 2002 I have been exploring SQL Server in many areas. I have been deep diving into various topics of SQL server having to do mainly with: ***administration***, ***development/data access*** and ***performance tuning***. These are the three areas of SQL Server I like the most and on which I base, but not limit, my interaction with SQL Server and acquisition of knowledge. To this end, I have decided to publish three eBooks on the three above mentioned areas of SQL Server.

The content of the eBooks is mainly based on articles that I have already published on my blog. It is actually a collection of those articles, edited and enriched with additional content. This is the reason the series is called **"The SQL Server and .NET Blog" eBook Series**. This is the second eBook and it is dedicated to SQL Server administration. Last but not least, under the title of each chapter, there is a URL pointing to the original article on my blog. By following the link, you will be able to leave comments for the articles in this eBook.

Who Should Read This Book?

This book is for database administrators and architects who monitor and administer SQL Server instances in order to keep them operating to the highest possible level of stability and performance. The book suggests several techniques that can be used for ensuring a healthy SQL Server instance. However, the book is not intended to be a step-by-step comprehensive guide. Additionally, it assumes at least intermediate-level experience with SQL Server administration.

Supported SQL Server Versions

Each article in this book has in the end an "***Applies to:***" note indicating the SQL Server versions against which the provided T-SQL scripts can be executed. There are also articles providing different T-SQL scripts for SQL Server 2000, SQL Server 2005 or later. Every time I write an article, I try to write code that can be executed in all versions of SQL Server unless stated otherwise. Even though SQL Server 2000 is not officially supported anymore, I am quite sure that still there are many of you out there who use this version of SQL Server for various reasons (i.e. application compatibility issues, etc.). My advice is to consider upgrading to a newer version of SQL Server the soonest possible, in order to be able to use the benefits of the latest SQL Server releases as well as being able to get official support and updates.

How Is This Book Organized?

This book is organized as follows. Chapter 1 discusses basic SQL Server maintenance tasks such as disk usage monitoring, history cleanup, backup-related topics, and more. Chapter 2 discusses security and compliance topics such as Policy-Based Management and encryption. Chapter 3 talks about different integration topics such as linked servers, use of proxies on SQL Server Agent job steps and Unicode support. Chapter 4 discusses special SQL Server topics like the Windows Internal Database (SSEE) and dynamic T-SQL generation. Chapter 5 discusses error handling and ways to overcome errors you might encounter due to external factors like permission issues, invalid user input, etc.

Upcoming Books of this Series

This is the second eBook of the series "*The SQL Server and .NET Blog*". The first book was published in October 2013 with the title "***Tuning SQL Server***". There is still one eBook to be published. The third eBook of this series will be dedicated to SQL Server development and data access and will be titled "*Developing SQL Server*". It will be released within 2014.

Feedback

I am a proud member of the worldwide SQL Server community. Feedback from you, my fellow community members, is more than welcome. Please feel free to visit the following link and provide your feedback:

http://sqlbooks.aartemiou.com/feedback

The survey is short. It will only take 5 minutes of your valuable time.

Acknowledgments

Writing a book is not an easy thing. It doesn't really matter the length of the book. Even if you just write a few pages, it takes a considerable amount of time and energy. In the case where you are not a professional technical writer but just a technical community guy like me, this amount of time is valuable "free" time after work, time from your family and beloved ones. You may use this time because you are just passionate in what you do. However, this is not enough, at least to me. Without the support of your family it just doesn't feel right. During this process I had all the support I needed. I am grateful for all the support my beautiful wife and daughter gave me during writing this book and for all their support and love in everything I do. Without their support and encouragement none of this would have been possible. This book is dedicated to them with all my love.

- Artemakis

Stay in Touch

Feel free to connect! You can find me on the following online channels:

- Twitter: http://twitter.com/artemakis
- The SQL Server and .NET Blog: http://aartemiou.blogspot.com/
- The SQL Server and .NET TV: http://www.youtube.com/user/sqlserverdotnetblog
- Artemakis Artemiou's SQL Server Books: http://sqlbooks.aartemiou.com
- Artemakis Artemiou's SQL Tools: http://sqltools.aartemiou.com
- Personal website: http://www.aartemiou.com/
- CodePlex: http://www.codeplex.com/site/users/view/ArtSQL

CHAPTER 1

Maintenance

IN THIS CHAPTER:

- Cleaning up Backup and Restore History Logs in MSDB
- Explicitly Control SQL Server Connections
- Backing up a Database in a Network Folder
- Backup Compression in SQL Server 2008
- Renaming Windows Logins in SQL Server
- The SQL Server Copy Database Wizard
- Getting the File Locations for all DBs in a SQL Server Instance
- Getting the Disk Usage Statistics for all Tables in a Database
- Retrieving Database File Sizes
- Retrieving Size Information for all Tables in a SQL Server Database
- Undocumented Stored Procedure sp_MSforeachdb

SQL Server is a very powerful data platform. However, in the real world, you cannot have a deployed data platform operating without maintaining it. When it comes to data platforms, and especially when they are deployed in large, enterprise environments, the "auto-pilot" is not an option.

One of the daily tasks of the Database Administrator (DBA) is to regularly run different tasks in order to maintain the good health of the installed SQL Server instances and thus the good health of the hosted databases. Such tasks can be: log monitoring and maintenance, disk usage statistics and handling of foreseen disk space issues, memory and CPU usage monitoring, database integrity checks, monitoring of database backups, fragmentation statistics/rebuild of indexes, and the list goes on.

SQL Server has a variety of internal mechanisms that ensure its proper operation, even in the absence of a suitable daily maintenance process. These mechanisms, among other, maintain data integrity. It is however, necessary to have a daily maintenance procedure in place, if you really want to have the healthiest possible and performant SQL Server instances, with the minimum possible bottlenecks on the different subsystems (i.e. I/O, network, CPU, memory, etc.).

This chapter includes selected articles from my blog having to do with performing different maintenance-related tasks on you SQL Server instances.

Cleaning up Backup and Restore History Logs in MSDB

(source: http://aartemiou.blogspot.com/2010/01/cleaning-up-backup-and-restore-history.html)

Sometimes, there are cases, where you might find out that the **MSDB** database is huge and needs to be shrinked again back to "normal" sizes.

One case which affects the size of MSDB, is the backup history. Every time a database backup/restore operation takes place in SQL Server, the relevant information is kept into certain system tables in the MSDB database.

SQL Server stores database backup and restore information into the following tables that exist in the MSDB database:

- **backupfile**
- **backupfilegroup**
- **restorefile**
- **restorefilegroup**
- **restorehistory**
- **backupset**

The following queries retrieve information regarding the backup and restore operations that took place on a specific database:

```
--This query returns information about the backup set
SELECT *
FROM msdb.dbo.backupset
WHERE [database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.1: Retrieving backup set information.

```
--This query returns information about the backup files including
--the logical and physical file names, drive, etc.
SELECT bf.*
FROM msdb.dbo.backupfile bf
INNER JOIN msdb.dbo.backupset bs
ON bf.[backup_set_id]=bs.[backup_set_id]
AND bs.[database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.2: Retrieving backup file information

```
--This query returns information about the filegroups that were backed up.
SELECT bg.*
FROM msdb.dbo.backupfilegroup bg
INNER JOIN msdb.dbo.backupset bs
ON bg.[backup_set_id]=bs.[backup_set_id]
AND bs.[database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.3: Retrieving information about the backup filegroups.

```
--This query returns information about when a database was restored.
SELECT *
FROM msdb.dbo.restorehistory
WHERE [destination_database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.4: Retrieving information about the restore history of a database.

```
--This query returns information about the physical files involved in
--the restoration process.
SELECT rf.*
FROM msdb.dbo.restorefile rf
INNER JOIN msdb.dbo.restorehistory rh
ON rf.[restore_history_id]=rh.[restore_history_id]
AND rh.[destination_database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.5: Retrieving information about physical files involved in a restore.

```
-- This query returns information about the restored filegroups.
SELECT rg.*
FROM msdb.dbo.restorefilegroup rg
INNER JOIN msdb.dbo.restorehistory rh
ON rg.[restore_history_id]=rh.[restore_history_id]
AND rh.[destination_database_name]='Enter_DB_Name_Here';
GO
```

Listing 1.6: Retrieving information about the restored filegroups.

In the case where the MSDB database is really large and you want to reduce its size, you can try removing any unnecessary backup and restore history logs.

Instead of manually deleting these logs from the above tables, you can use a SQL Server system stored procedure instead.

This stored procedure is called "**sp_delete_backuphistory**" and exists in the MSDB system database.

The usage for the above stored procedure is:

```
sp_delete_backuphistory [ @oldest_date = ] 'oldest_date';
GO
```

Listing 1.7: Deleting backup history.

Example:

```
exec msdb.dbo.sp_delete_backuphistory '2010-01-01';
GO
```

Listing 1.8: Example of deleting backup history.

The above command will clean up the backup and restore history logs **up to 01/01/2010**.

For more information you can visit SQL Server Books Online.

➔ ***Applies to: SQL Server 2000 (partially), 2005 or later (fully).***

▪ Explicitly Control SQL Server Connections

(source: http://aartemiou.blogspot.com/2009/04/explicitly-control-sql-server.html)

There is many times where you might want to explicitly control the connections to an instance of SQL Server and more specifically to a database.

Such scenarios might include:

- A restore procedure of a backup set which cannot complete as there are still active connections to the specific database.
- You need to drop a database and you cannot do it as there are still active connections to that database.
- … and generally when you want to perform an operation on a database which requires a single user mode but there are other active connections as well and you want to terminate them.

Even though there is more than one method for handling the active connections to a database, here I will present one that I consider very efficient as it allows the DBA to get information about all the active connections to a specific database and take direct actions.

The key table in this case is **sysprocesses** which exists in the **master** database. Sysprocesses has a separate row for each unique **SPID** in a SQL Server connection. A SQL Server connection might be among other:

- A query window connected to a specific database
- Reporting services
- Analysis Services
- Management Studio
- Any other database application (client) connected to a database.

The question here is: *how can the **sysprocesses** table help us explicitly terminate a SQL Server connection?*

By issuing a query against the sysprocesses table, we are able to find all the connections to a given database, get their SPIDs and then by using the SQL Server command named *KILL*, forcibly terminate these connections.

For example, consider that we have a database called **TestDB**. On this database, let's say that we have performed a task with SSMS a while ago (i.e. Select Top 1000 Rows) and we have also opened a query window that it is connected to that database.

For getting the SPIDs along with some other useful information we execute the following query:

```
SELECT spid,login_time,program_name,nt_domain,nt_username
FROM master..sysprocesses
WHERE DBID=DB_ID('TestDB');
GO
```

Listing 1.9: Retrieving the connections on a database.

And here are the results of the query:

spid	login_time	program_name	nt_domain	nt_username
51	2009-04-09 21:33:49.193	Microsoft SQL Server Management Studio	TESTING	User1
53	2009-04-09 23:01:17.050	Microsoft SQL Server Management Studio - Query	TESTING	User1

So, what do the above results mean? Well, in plain words we can see that there are two open connections to the **TestDB** database. The connection with **SPID 51** comes from the SQL Server Management Studio (SSMS) and connection with **SPID 53** is actually a Query window connected to the database. Also, the **login_time** displays the exact time each connection was established, the **program_name** displays the application or SQL Server module/tool which uses the connection, the **nt_domain** displays the Windows domain for the database client and the **nt_username** is the Windows user name used by the client (in the case where Windows Authentication is used).

Sysprocesses contains much more information than the columns used in the above query. Such information can be: memory usage information, disk I/O information, transactional information, network information, etc. To this end, besides of using sysprocesses for finding an SPID, you can also use it for performance tuning and other operations as it can provide significant information about the current usage of the SQL Server instance and specific databases.

MSDN Documentation for the **sysprocesses** table in SQL Server 2000 can be found here, for SQL Server 2005 here, and for SQL Server 2012 and 2008/R2 here.

And now back to our case scenario. If we are in a scenario like the ones described in the beginning of this article, or in a similar case where we want to terminate active SQL Server

connections that prevent us from executing another task (in this case SPIDs 51 and 53), then an option is to first find these SPIDs (using the **sysprocesses** table) and then issue the SQL Server KILL command.

So, now that we have the SPIDs of the two connections, we can terminate them with the following statements:

```
KILL 51;
GO
KILL 53;
GO
```

Listing 1.10: Terminating user processes.

Important: After terminating the connections-SPIDs, you cannot use the specific "programs" associated with those SPIDs (i.e. Query window) because the connections do not exist anymore. In such case you have to re-initiate the connection. If you are for example in a Query window and you try to execute a T-SQL script, the first time you try to execute it you will get the following error message:

"A transport-level error has occurred when sending the request to the server. (provider: Shared Memory Provider, error: 0 - No process is on the other end of the pipe.)"

The above message means that the connection/process was ended. If you try again, a new connection will be established and you will be able to proceed.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Backing up a Database in a Network Folder

(source: http://aartemiou.blogspot.com/2011/08/backing-up-database-in-network-folder.html)

This article discusses about a quite simple task in SQL Server: **backing up a database in a network folder / remote server**.

Let's consider the following scenario:

You have a SQL Server Instance located on a server with only one local drive (**!**) and you urgently need to backup a database somewhere! Well, the first thing that comes on my mind, is to backup the database on a remote location (in this scenario take into consideration that would do not have physical access to the server and cannot mount a USB flash drive).

In order to backup the database in the above scenario you first need to mount the remote location (network folder) as a backup device.

To this end, let's say you have the network folder **\\serverName\backupFolder** and you want to mount it as a backup device. For doing that, you have to run the following stored procedure:

```
USE master;
GO

EXEC sp_addumpdevice
'disk','NetWorkDeviceName','\\serverName\backupFolder\BackupFileName.bak';
GO
```

Listing 1.11: Mounting a network folder as a backup device.

Then from SSMS you will be able to select the mounted backup device as the backup location.

However, there is one consideration; **the service account which runs the SQL Server Instance needs to have read/write permissions on the network folder (shared permissions)**.

In the opposite case you will receive the following "access denied" error message:

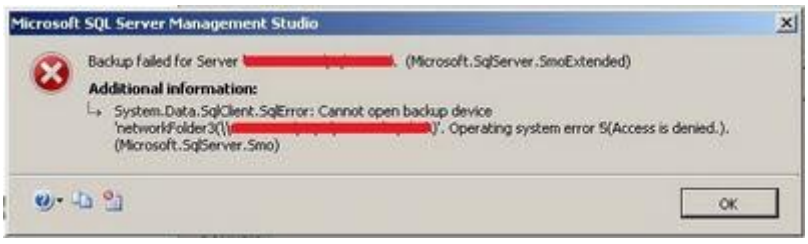


Figure 1.1: Access denied error message while trying to place a backup set on a network location.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Backup Compression in SQL Server 2008

(source: http://aartemiou.blogspot.com/2008/12/backup-compression-in-sql-server-2008.html)

SQL Server 2008 introduced **Backup Compression**. This enhancement allows the user when backing up a database to activate backup compression which is performed on the fly.

To do this you have to right-click on a database, select **Tasks** and then **Backup**. In the Back Up Database dialog which pops up, if you go to the **Options** tab, you will notice that there is a feature on the bottom of the dialog which says **Compression**. Then you can set the backup compression for the current database. To this end you are presented with the following options:

1. Use the server default setting
2. Compress backup
3. Do not compress backup

If you select the **Compress backup** or **Use the server default setting** and your instance of SQL Server is set up to compress the backups by default, when you initiate the backup process the backup set will be automatically compressed thus achieving high compression rations and

reducing the required storage.

Note: If you have performed a backup operation on the database before and you have not used the backup compression option but now you are trying to use it by appending a new backup to the existing set you will get an error. The same stands for the opposite case. A backup set can either contain only compressed database backups or only non-compressed database backups but never both types. The reason for this is that by the time you have a non-compressed backup set, you cannot add a compressed backup to this and vice versa.

One solution to the above restriction is to use the option **Backup to a new media set and erase all existing backup sets**. This will allow you to proceed with the backup but it will also erase the previous backup sets for the specific database. To this end, it is always better to decide the backup method to be followed for each database before you start taking backups of it.

➔ ***Applies to: SQL Server 2008 or later.***

▪ Renaming Windows Logins in SQL Server

(source: http://aartemiou.blogspot.com/2009/05/renaming-windows-logins-in-sql-server.html)

When a DBA creates Windows Logins in SQL Server, the format of these logins is:

[Domain or Server Name]\[Windows Username]

[Domain] can also be the local Server name if the user is a local Windows User.

In the case where the Server name changes and the Windows Login names remain the same in SQL Server then the login will not be usable. To this end, these login names have to be renamed. The syntax for renaming Windows Logins in SQL Server is:

```
ALTER LOGIN "[Domain or Server Name]\[Windows Username]"
WITH NAME="[New Domain or New Server Name]\[Windows Username]";
GO
```

Listing 1.12: Renaming a Windows login.

*Note: You have to use the double quotes.

If you want to rename a SQL Server login (SQL Server Authentication) you can use one of the following syntaxes:

Syntax 1:

```
ALTER LOGIN "[SQL Server Login Name]"
WITH NAME="[New SQL Server Login Name]";
GO
```

Listing 1.13: Renaming a SQL Server login – Method 1

Syntax 2:

```
ALTER LOGIN [SQL Server Login Name]
WITH NAME=[New SQL Server Login Name];
GO
```

Listing 1.14: Renaming a SQL Server login – Method 2

As you may noticed from the above two syntaxes, you can either select to use or not double quotes.

➔ ***Applies to: SQL Server 2005 or later.***

▪ The SQL Server Copy Database Wizard

(source: http://aartemiou.blogspot.com/2009/04/sql-server-copy-database-wizard.html)

There is many times where you might need to copy or move a database with the least amount of effort. The "manual" way of performing the above tasks is by using the backup/restore or attach/detach functionality.

SQL Server 2000 introduced the **Copy Database Wizard;** a feature which allows the DBA with just a few clicks to copy or move an entire database. Though, this version of the Copy Database Wizard allowed copying/moving databases only between **different** instances of SQL Server 2000 (it also allowed copying/moving databases from SQL Server 7.0 instances to SQL Server 2000).

In more recent versions of SQL Server, like 2005, 2008/R2 and 2012, the Copy Database Wizard was enhanced with even more functionality. An example is that now you can also copy a database within the same SQL Server instance, thus easily creating a duplicate of it, by using the wizard.

In this article we will go through the process of using the Copy Database Wizard in SQL Server 2008.

In order to start the wizard (in SQL Server 2005, 2008/R2 **and 2012(?)**), you have to right-click on a database from within SQL Server Management Studio and from **Tasks** select the **Copy Database** task (in SQL Server 2000 you can run the wizard by clicking once on the registered SQL Server instance in Enterprise Manager, then click on the "Tools" menu, select "Wizards", and select "Copy Database Wizard" in the category of "Management" wizards). Then you will be presented with the welcome screen of the Copy Database Wizard:

Figure 1.2: Copy Database Wizard – First Screen

Click **Next**.

Then you have to specify the **Source Server** and the **Authentication Method** for accessing it**:**

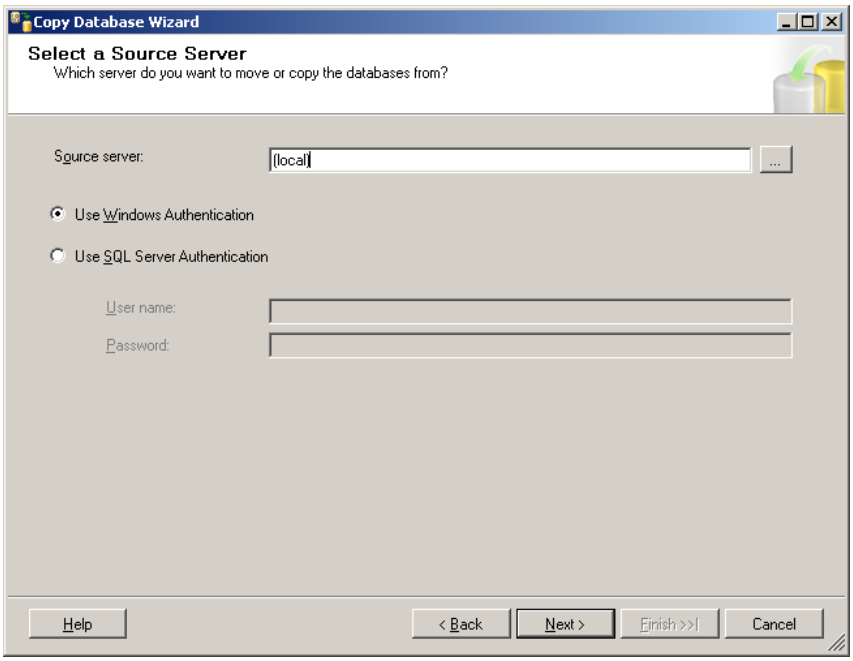


Figure 1.3: Copy Database Wizard – Specify source server and authentication type

Click **Next**.

This screen allows you to choose the **Transfer Method** based on which the wizard will perform the copy/move database task:

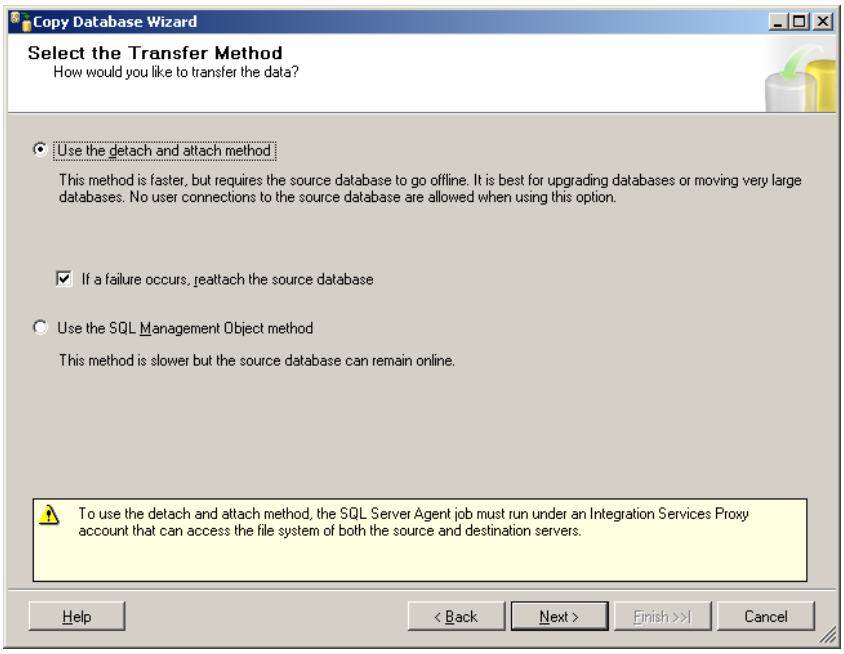


Figure 1.4: Copy Database Wizard – Select the transfer method

Click **Next**.

So here you are presented with two transfer methods: (i) Use the detach and attach method, and (ii) Use the SQL Management Object Method (SMO). The first method requires the database to go offline (that is, downtime) while the second requires no database downtime but it is slower than the first one. After you select a Transfer Method click on **Next**.

You are then presented with a screen containing all the databases on the Source Server you specified earlier, where you can select which of those databases you want to Move or Copy.

Note that even though you are presented with the SQL Server system databases on this screen as well, you cannot copy or move them. For this example let's use the **TestDB** database to copy it locally on the same SQL Server instance. To this end, we select the **Copy** checkbox for this database (the procedure is the same for the **Move** option with the difference that the Wizard will delete the database from the Source Server after moving it to the Destination):

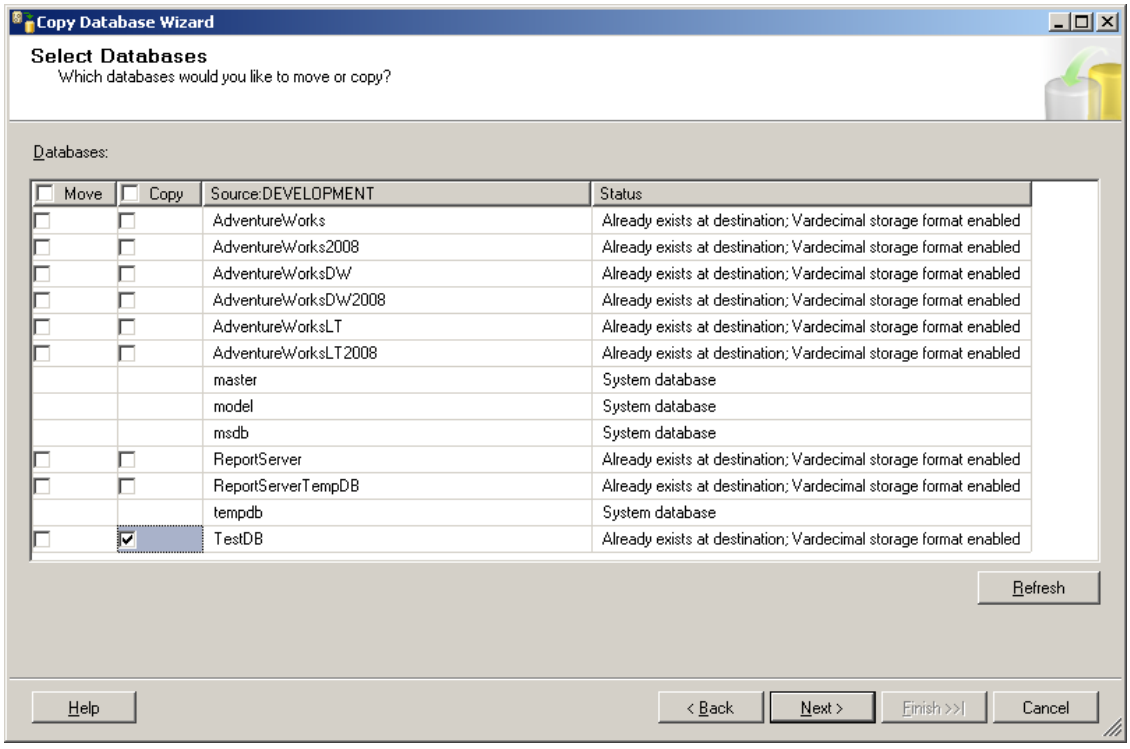


Figure 1.5: Copy Database Wizard – Select the databases to move or copy.

Click **Next**.

The next screen allows you to configure the destination database. To this end, you can specify the new database's name, the database files paths and names, and also instruct the wizard what to do in the case the destination database already exists:

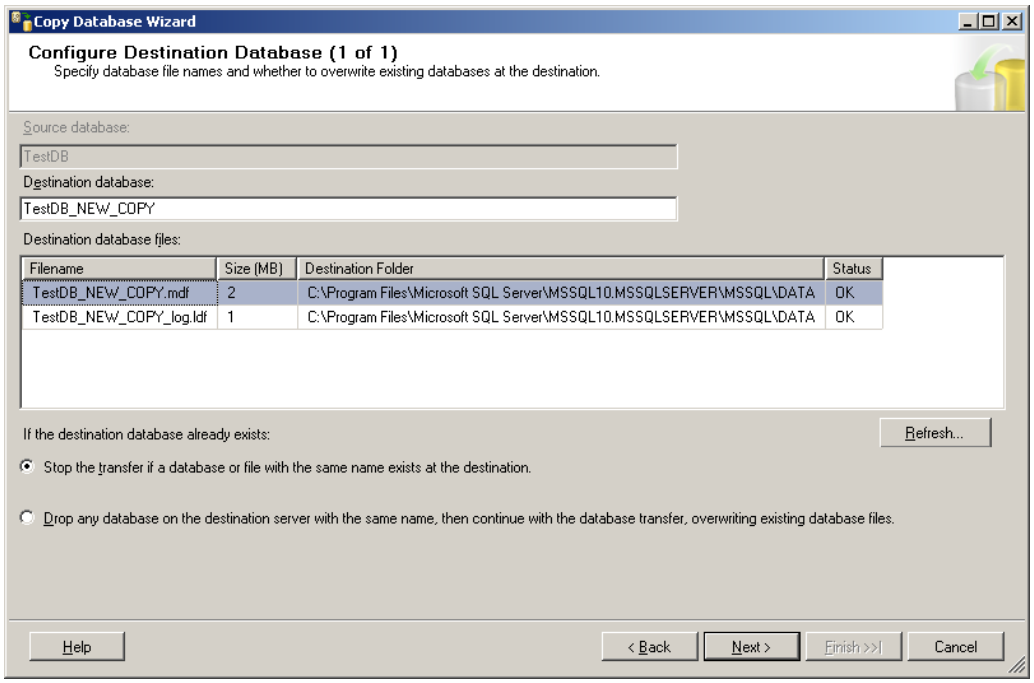


Figure 1.6: Copy Database Wizard – Configure the destination database.

Click **Next**.

So, what does actually the Copy Database Wizard do? It receives all the necessary parameters by the user, and it builds a SSIS Package! On the following screen you can configure the package:

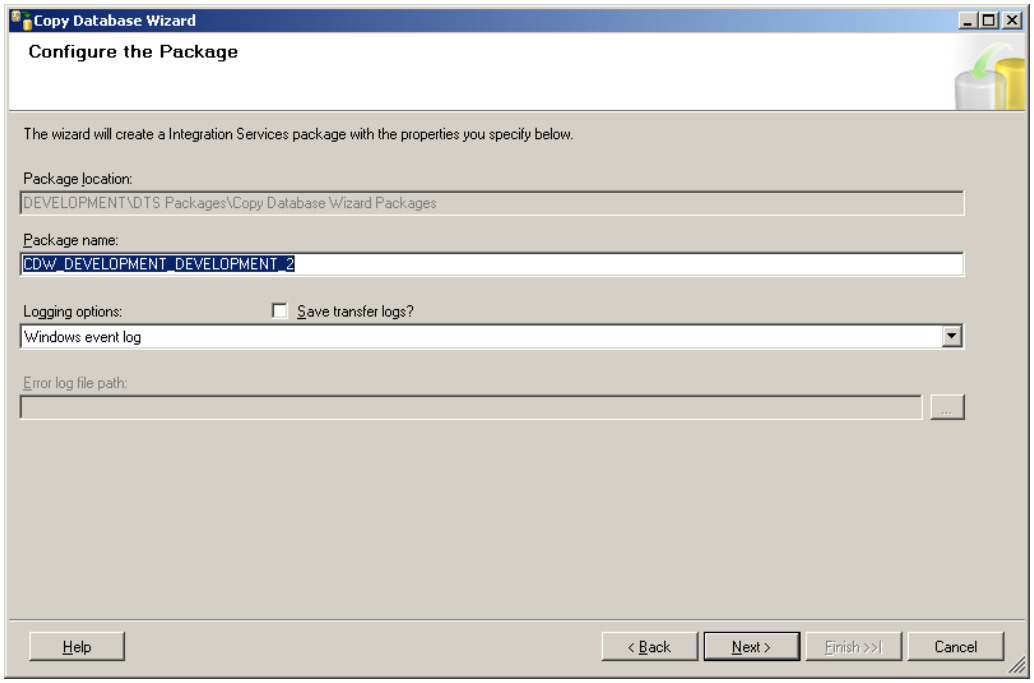


Figure 1.7: Copy Database Wizard – Configure SSIS package that will be created.

Click **Next**.

Last but not least, in another screen, you are presented with the option of when to run the SSIS Package. You can either run it immediately or schedule it to run later. Also on this screen, you must select an **Integration Services Proxy account** which will allow SQL Server Agent to access and run the SSIS Package:

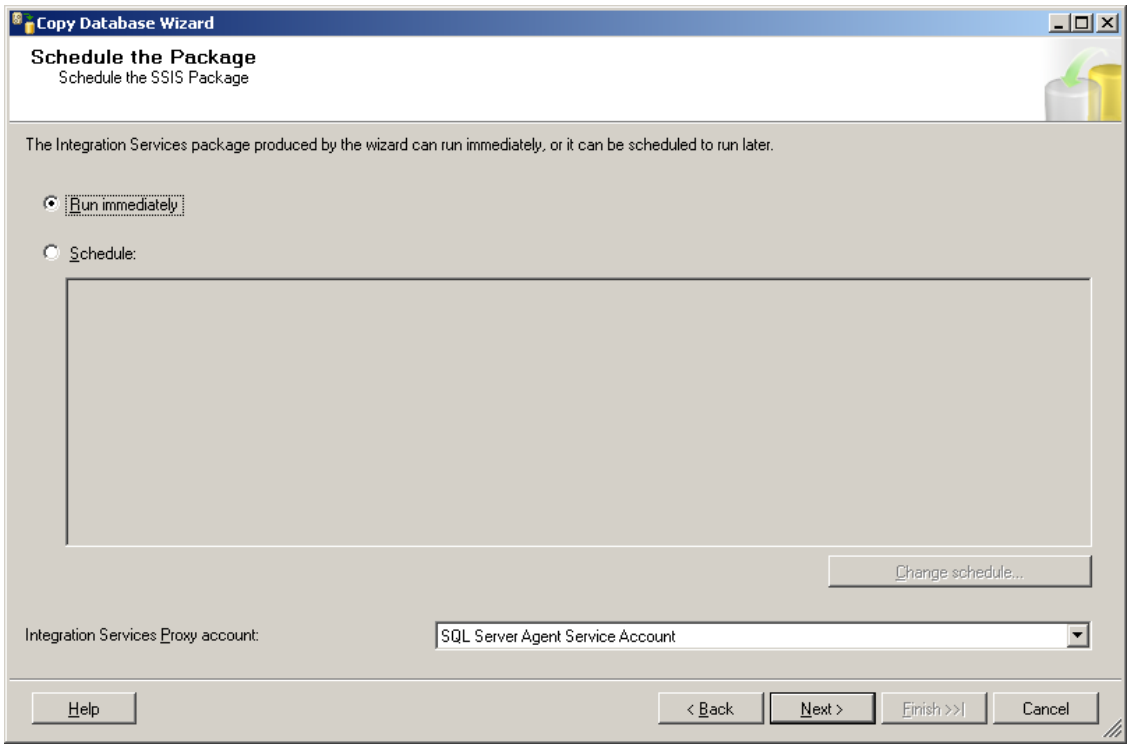


Figure 1.8: Copy Database Wizard – Schedule when the SSIS package will be executed.

Note: The SQL Server Agent service must be running for being able to execute the SSIS Package created by the Copy Database Wizard.

Then by clicking on **Next** you are presented with one last screen which contains a summary of the tasks the Copy Database Wizard has to do. You click on **Finish** and the process will run.

Considerations

While using the Copy Database Wizard in SQL Server 2005 or later, you must take into account some considerations. Some of them are:

- You must be a member of the sysadmin fixed server role on both the Source and Destination Database Servers.
- Your SQL Server installation should include SQL Server 2005 Integration Services (SSIS) or later.

- When you choose to Move a database, the Wizard deletes it from the Source Server. When you choose to Copy a database, the wizard does not delete it from the Source Server.

  For comprehensive documentation regarding the Copy Database Wizard you can visit this link.

➔ ***Applies to: SQL Server 2000 (partially), 2005 or later (fully).***

▪ Getting the File Locations for all DBs in a SQL Server Instance

(source: http://aartemiou.blogspot.com/2012/12/t-sql-tip-get-file-locations-for-all.html)

This article presents how you can dynamically build T-SQL statements for various purposes. An example is generating T-SQL statements for getting the file locations for all DBs in a SQL Server instance.

The query is provided below:

```
--
-- Dynamically builds  T-SQL statements for retrieving the file
-- locations for all the databases in the SQL Server Instance
--
-- SQL Server versions  supported: SQL Server 2005 or later
--
SELECT 'use '+ [name]+'; select '''+[name]+''' as DBName,cast ([name] as
varchar(45)) as LogicalFileName,cast (filename as varchar(100)) as FileName
from sysfiles;' as SQLStatement
FROM master.sys.databases;
GO
```

Listing 1.15: Retrieving the file locations of all DBs by generating a set of T-SQL statements.

Details: Just execute the statements generated from the above query and you will receive the file locations for all the databases in the SQL Server instance.

If you don't want to dynamically generate T-SQL statements for getting the physical file locations for all databases, you can always use the following T-SQL statements:

SQL Server 2000

```
SELECT DB_NAME(dbid), name, [filename]
FROM master.dbo.sysaltfiles;
GO
```

Listing 1.16: Retrieving the DB file locations in SQL Server 2000.

SQL Server 2005 or later

```
SELECT DB_NAME(database_id), name, physical_name
FROM master.sys.master_files;
GO
```

Listing 1.17: Retrieving the DB file locations in SQL Server 2005 or later.

For more info, check out the following links:

- sys.sysfiles
- sys.databases

➔ ***Applies to: SQL Server 2000 or later.***

▪ Getting the Disk Usage Statistics for all Tables in a Database

(source: http://aartemiou.blogspot.com/2012/08/getting-disk-usage-statistics-for-all.html)

Even though there are standard reports and other GUI tools for doing this task in SQL Server 2005 or later, there is an alternative way by using T-SQL along with the stored procedure sp_spaceused and the undocumented stored procedure **sp_msforeachtable.** Here's how:

```
--Select the database to be scanned for table disk usage
USE [DATABASE_NAME];
GO

--Create temporary table 1 - Sizes will be in strings/KB
create table #tmpSizes(
[name] nvarchar(200),
[rows] varchar(50),
reserved varchar(50),
data varchar(50),
index_size varchar(50),
unused varchar(50)
);
GO

--Create temporary table 1 - Sizes will be in KB
create table #tmpSizesFinalKB(
[name] nvarchar(200),
[rows] int,
reserved int,
data int,
index_size int,
unused int
);
GO

--continues on next page…
```

```
--…from previous page

--Get the disk usage per table and store in temp table 1
INSERT INTO #tmpSizes
EXEC sp_MSforeachtable @command1="sp_spaceused '?'";
GO

--Indirect casting and copying of the information in temp table 2
--This is only when you want to store the disk usage statistics
--in a form that allows sorting operations etc.
insert into #tmpSizesFinalKB
select
[name],
replace([rows],'KB',''),
replace(reserved,'KB',''),
replace(data,'KB',''),
replace(index_size,'KB',''),
replace(unused,'KB','')
from #tmpSizes;
GO

--Access the disk usage results (in KB)
select *
from #tmpSizesFinalKB
order by reserved desc;
GO
```

Listing 1.18: Getting the Disk Usage Statistics for all Tables in a Database.

Now you can easily manipulate the disk usage statistics in temp table #tmpSizesFinalKB in order to represent them in the form you may like.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Retrieving Database File Sizes

(source: http://aartemiou.blogspot.com/2012/08/t-sql-tip-retrieving-database-file-sizes.html)

The following script generates T-SQL statements that when ran, they return file size information for all user databases on a SQL Server instance.

```
--Script that generates T-SQL providing size information for
--all database files on a SQL Server Instance
SELECT 'SELECT '''+[name]+''' as DBName,cast(f.name as varchar(25)) as
DBFileName,f.Filename as PhysicalFileName,cast (round((f.size*8)/1024.0,2)
as decimal(18,2)) as FileSizeinMB FROM '+[name]+'..SYSFILES f'
FROM SYS.SYSDATABASES
WHERE [name] not in
(
'master',
'tempdb',
'model',
'msdb'
);
GO
```

Listing 1.19: Generating a set of T-SQL statements for retrieving size information for user DB files.

For more info, check out the following MSDN links:

- sys.sysdatabases
- sys.sysfiles

➔ ***Applies to: SQL Server 2005 or later.***

▪ Retrieving Size Information for all Tables in a SQL Server Database

(source: http://aartemiou.blogspot.com/2012/02/how-to-retrieve-size-information-for.html)

In an article in the first eBook of this series, on SQL Server Tuning, I described how you can rebuild all the indexes of a database in SQL Server by making use of the undocumented stored procedure "sp_MSforeachtable".

Another common task is when you want to retrieve size information for all the tables in a database.

Again, by using "**sp_MSforeachtable**", you can easily do that in three simple steps:

```
--Step 1:
--Create temporaty table for the session
create table #tblInfo(
[name] nvarchar (255),
rows int,
reserved varchar(100),
data varchar(100),
index_size varchar(100),
unused varchar(100)
);
GO

--Step 2:
--Using the stored procedure sp_spaceused retrieve
--the size information for all tables and store it in the temporary table
EXEC sp_MSforeachtable @command1="INSERT #tblInfo EXEC sp_spaceused '?'";
GO

--Step 3:
--Access the results
select * from #tblInfo;
GO
```

Listing 1.20: Retrieving size information for all the tables in a database.

Note: Because **sp_spaceused** returns the size information as a string (except the number of rows), you will have to manipulate the data in the temporary table using casting prior to run sorting operations etc.

Examples:

```
--Sort the results by unused space (descending)
SELECT * FROM #tblInfo
ORDER BY cast(substring(unused,0,charindex(' ',unused)) as int) DESC;
GO

--Sort the results by reserved space (descending)
SELECT * FROM #tblInfo
ORDER BY cast(substring(reserved,0,charindex(' ',reserved)) as int) DESC;
GO
```

Listing 1.21: Casting the size information of all tables to the int data type.

Additionally, you can create another temporary table which can contain the converted values (i.e. in MB instead of KB) of the first table.

For examnple:

```
SELECT [name],
[rows],
cast(substring(reserved,0,charindex(' ',reserved)) as int)/1024 as reserved_in_MB,
cast(substring(data,0,charindex(' ',data)) as int)/1024 as data_in_MB,
cast(substring(index_size,0,charindex(' ',index_size)) as int)/1024 as
index_size_in_MB,
cast(substring(unused,0,charindex(' ',unused)) as int)/1024 as unused_in_MB
INTO #tblInfoConverted
FROM #tblInfo;
GO

-- and this now you can retrieve the converted information.
SELECT *
FROM #tblInfoConverted
ORDER BY reserved_in_MB desc;
GO
```

Listing 1.22: Further manipulation of the size information that was retrieved for all tables.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Undocumented Stored Procedure sp_MSforeachdb

(source: http://aartemiou.blogspot.com/2009/07/spmsforeachdb.html)

An interesting SQL Server stored procedure is **sp_MSforeachdb**. This stored procedure takes as parameters SQL commands which are then executed against **all** databases on the current SQL Server instance.

A simple example is the following which lists all the databases in the current SQL Server instance:

```
exec sp_MSforeachdb
@command1="print '?'";
GO
```

Listing 1.23: Listing all databases using sp_MSforachdb.

The stored procedure upon its execution replaces the question mark (**?**) with each database's name.

You can of course build more complex syntaxes like in the case where you want to set the compatibility level for **all** the databases to 110 (SQL Server 2012) after of course you ensured that the databases are fully compatible with this version of SQL Server:

```
exec sp_MSforeachdb
@command1="print '?'",
@command2="ALTER DATABASE [?] SET COMPATIBILITY_LEVEL = 110";
GO
```

Listing 1.24: Changing the compatibility level of all DBs to SQL Server 2012 (110).

***Note:** Even though the stored procedure **sp_MSforeachdb** enables you to massively execute SQL commands against all the databases in a SQL Server instance, you must be **extremely** careful with the commands which you compose, as this stored procedure access **all** the databases including the SQL Server system databases as well.

Also, I would not recommend executing T-SQL statements on a Production environment using SQL Server's undocumented stored procedures as one of the reasons for which they are undocumented, is that they might be removed at any time, as an example like in the case of a new SQL Server Service Pack or a new release.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Summary

In this chapter, you learned how to perform several maintenance tasks in different versions of SQL Server. Among other, you learned how to explicitly control connections to SQL Server instances, how to rename logins, how to retrieve different database-related information and how to set different backup options. The next chapter talks about one of the most important aspects of every software system that is Security. Security is one of the key areas not only generally in Information Technology but especially when it comes to Database Management Systems (DBMSs). The DBMS is the place where all data of the organization is stored and thus needs to be governed by strict security and data availability and protection policies. The next chapter discusses all these and presents many security-related features and techniques that can be used in SQL Server in order to protect your data.

CHAPTER 2

Security

IN THIS CHAPTER:

- Policy-Based Management in SQL Server 2008
- Migrating to a Contained Database in SQL Server 2012
- Changing the Database Owner in a SQL Server Database
- Creating Logins for Orphaned SQL Server Users
- Transparent Data Encryption in SQL Server 2008
- Encrypting SQL Server Databases
- Retrieving Security-Related Info for SQL Server Logins
- Security Changes in SQL Server 2008
- The SELECT ALL USER SECURABLES Permission in SQL Server 2014

Data Security has always been one of the key areas of Information Technology. By the time data is stored electronically, they are automatically exposed to the dangers of unauthorized access and unwanted actions. Many international standards, laws and policies exist in order to enforce procedures having to do with data security.

Some data security technologies are: file system-level encryption, hardware-based cryptography (i.e. HSMs), data backup, authentication, authorization, etc.

SQL Server provides a variety of built-in technologies that support data security to the maximum possible level. These technologies are:

- Windows and Mixed mode Authentication Modes.
- Server and Database roles.
- Ownership and User-Schema Separation.
- Authorization and Permissions.
- Data Encryption (keys, encryption algorithms, transparent data encryption).
- Endpoints.
- Various Backup Methods.
- Policy-Based Management.
- Surface Area Configuration.

All the above technologies, when efficiently combined, can provide a very high level of security for your SQL Server instances.

This chapter includes selected articles from my blog having to do with data security in SQL Server and how you can make use of different technologies that are built-in in SQL Server in order to rise up the level of security in your SQL Server instance.

Policy-Based Management in SQL Server 2008

(source: http://aartemiou.blogspot.com/2010/04/policy-based-management-in-sql-server.html)

Policy-Based Management is indeed one of the greatest administration features in SQL Server 2008. With this feature you can easily manage one or more instances of SQL Server and apply different "policies" in order to make your SQL Server instance compliant to your organization's data governance policies.

The main concept in Policy-Based Management is that Policy Administrators can set up policies that are evaluated upon certain events within the scope of the monitored SQL Server instance(s). After setting up the policies, based on certain scenarios, the policies are evaluated and specific actions are automatically performed as set up in the active Policies.

To this end, there are four evaluation modes:

- **On demand** - This mode evaluates the policies when explicitly requested by the user.
- **On change: prevent** - This mode uses DDL triggers in order to prevent Policy violations.
- **On change: log only** - This mode allows Policy violations but it logs each violation using event notification. It logs the violations to the SQL Server logs and Windows Application logs.
- **On schedule** - This mode allows the user to schedule (using a SQL Server Agent job) when the policies will be evaluated.

Let's get a first look and feel at the Policy-Based Management set up dialogs in SSMS. In the following screenshot you can see the Policy Management module in the Object Explorer in SSMS:

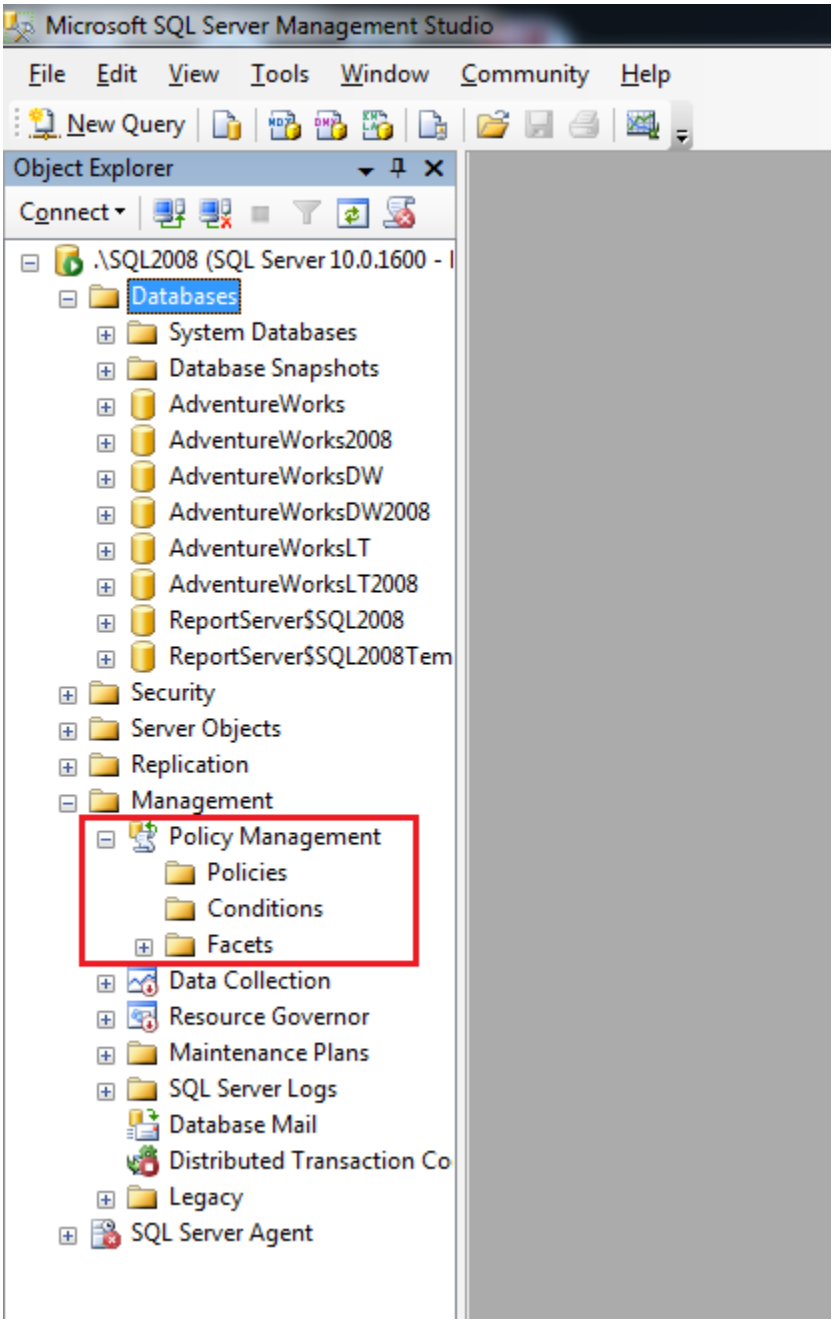


Figure 2.1: The Policy Management Module in SSMS.

The module provides the following functionality:

- When right-clicking on Policies you can create a new Policy.
- When right-clicking on Conditions you can create a new condition and then assign it to an existing or a new Policy.
- The Facets is a collection of SQL Server objects that allow you to set up conditions and policies against their properties. By right-clicking on a Facet you can create a condition or a Policy.

Note that the Policy Management module falls under Management as its scope is for the entire SQL Server instance. Of course, via the Policies you create, you can evaluate multiple SQL Server instances, certain databases, even specific database objects.

Now let's proceed with an example, showing how we can set up policies based on the abovementioned evaluation modes. For this example, we will be using the "AdventureWorks2008" sample database.

On demand

We right-click on Policies and we select **New Policy**. We are presented with the following dialog where we provide a name for the Policy and then we either select the check condition or we create a new one. In this case we will be creating a new condition:

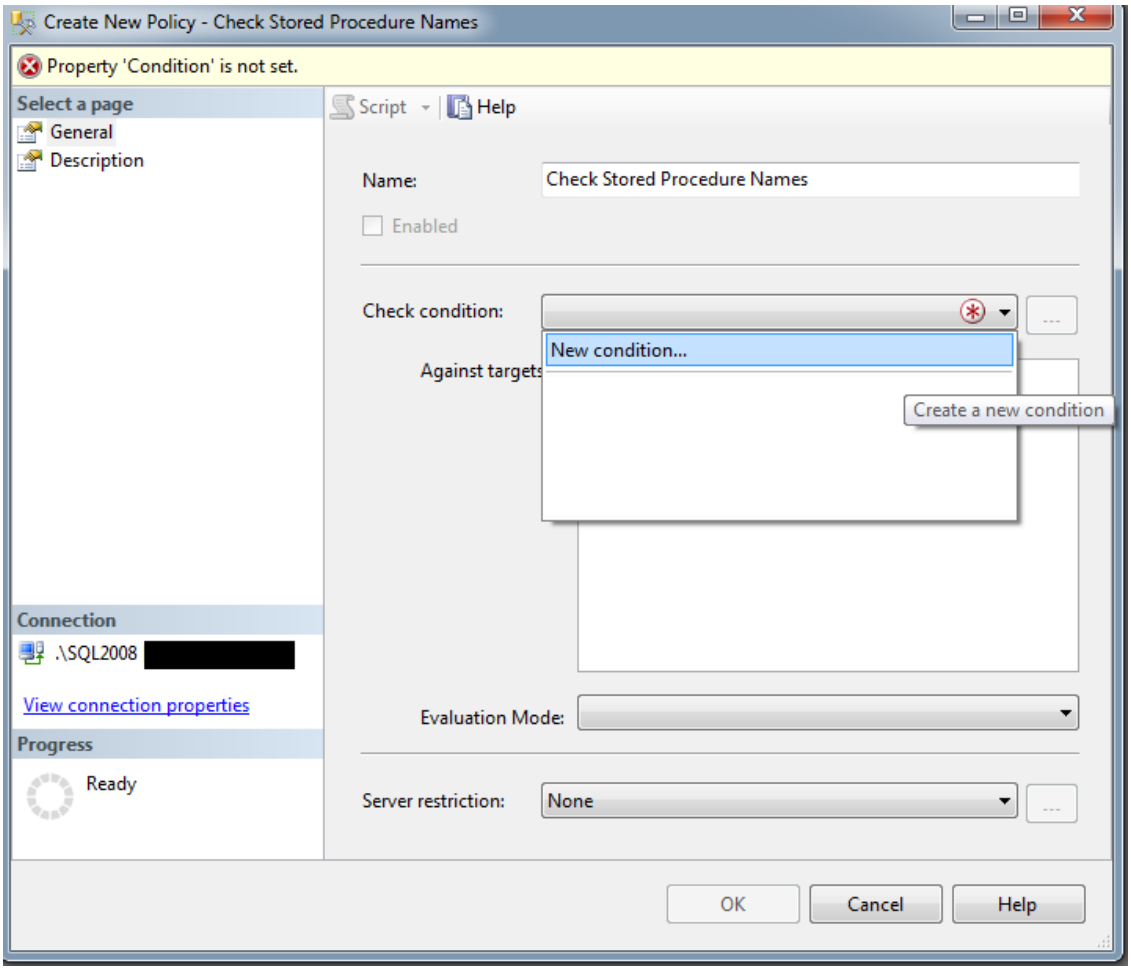


Figure 2.2: Policy-Based Management - Creating a new Policy.

The next screenshot shows the newly created condition:

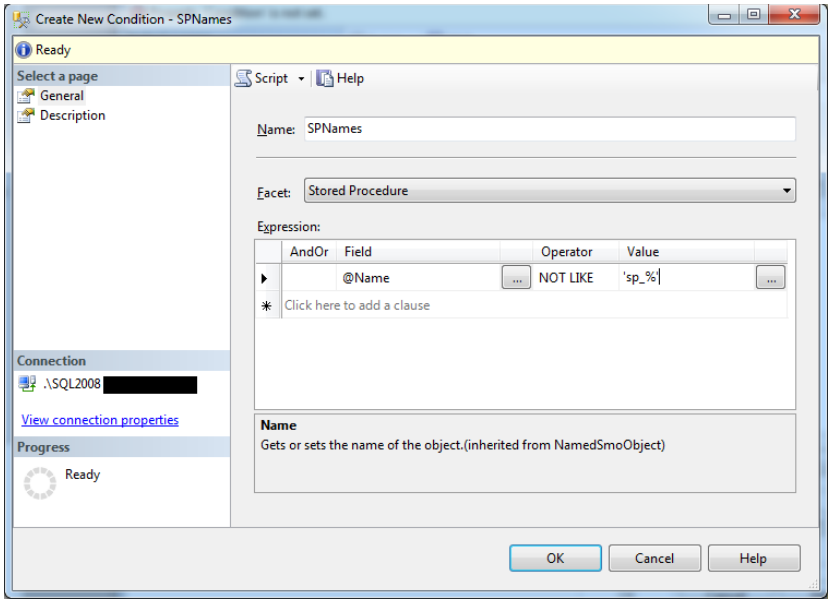


Figure 2.3: Policy-Based Management - Creating a new condition.

For creating a new condition, you must provide a name, select the Facet (in this example the **Stored Procedure**), and set the expressions for evaluating the properties you want. In this case we want the condition to track all the stored procedures that their names **start** with **'sp_'**. We click on the **OK** button and we are returned to the Policy:

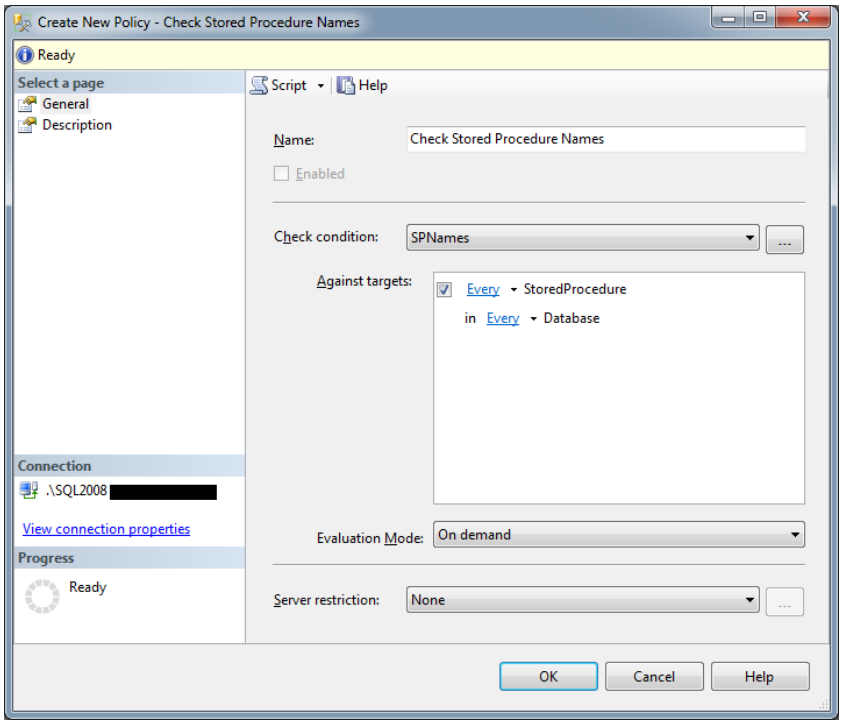


Figure 2.4: Policy-Based Management – Setting the Condition for a Policy.

You can see that the Policy now contains the "**SPNames"** condition we created in the previous step. Also, we set up the Evaluation Mode to "**On Demand"**. Note that this type of condition allows you to select any Evaluation Mode.

The available Evaluation Modes are dependent on the type of the condition. For example there are conditions for which a DDL trigger cannot be executed so the **"On change: prevent"** mode cannot be selected in such case, etc.

Also, in the **"Against targets"** list you can target specific databases against which the condition will be evaluated. Additionally, in the **"Server restriction"** you can select against which SQL Server instances the condition will be executed. **Remember**: Policy-Based Management allows managing multiple instances of SQL Server.

Now, let's set the Policy to only evaluate the **"AdventureWorks2008"** database by clicking on the "In **Every** - Database" list item. Here's the new condition for restricting the Policy's evaluation on the specific database:

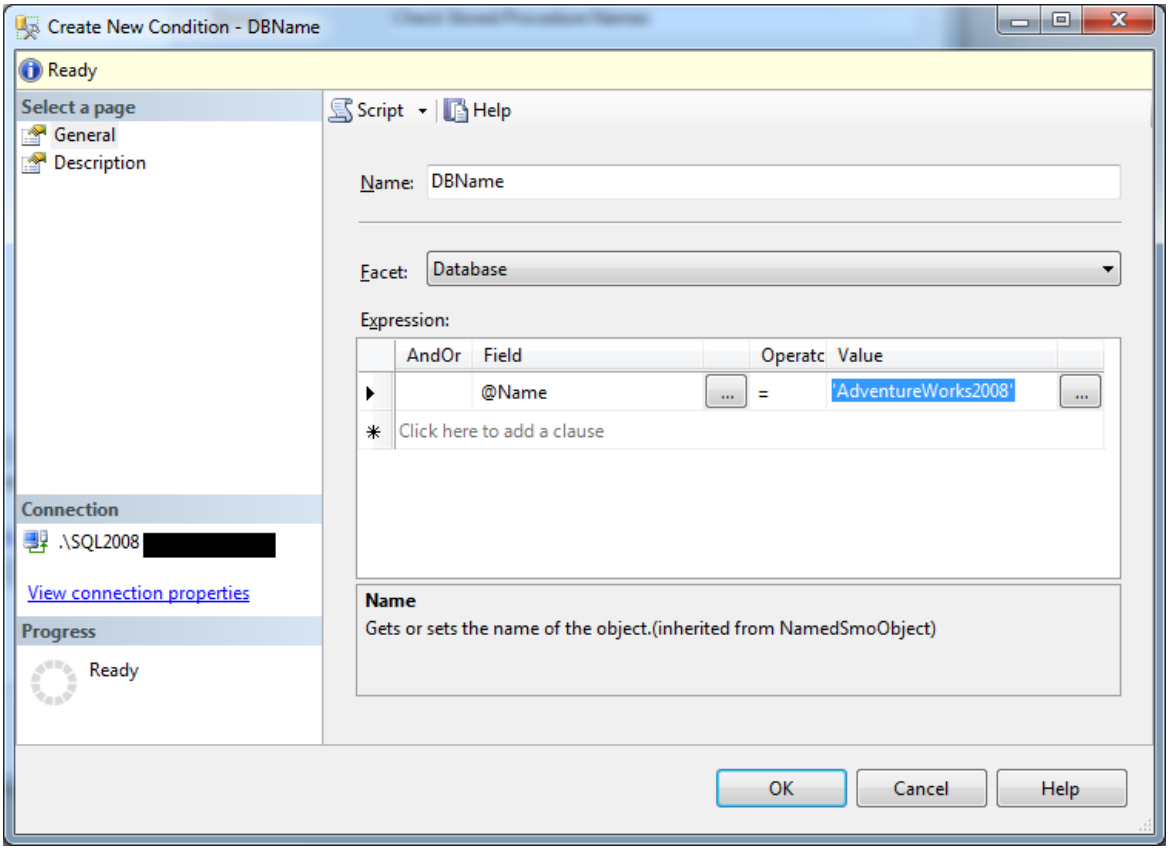


Figure 2.5: Policy-Based Management – Creating a new Condition.

We click on **OK** and we are returned back to the Policy properties dialog:

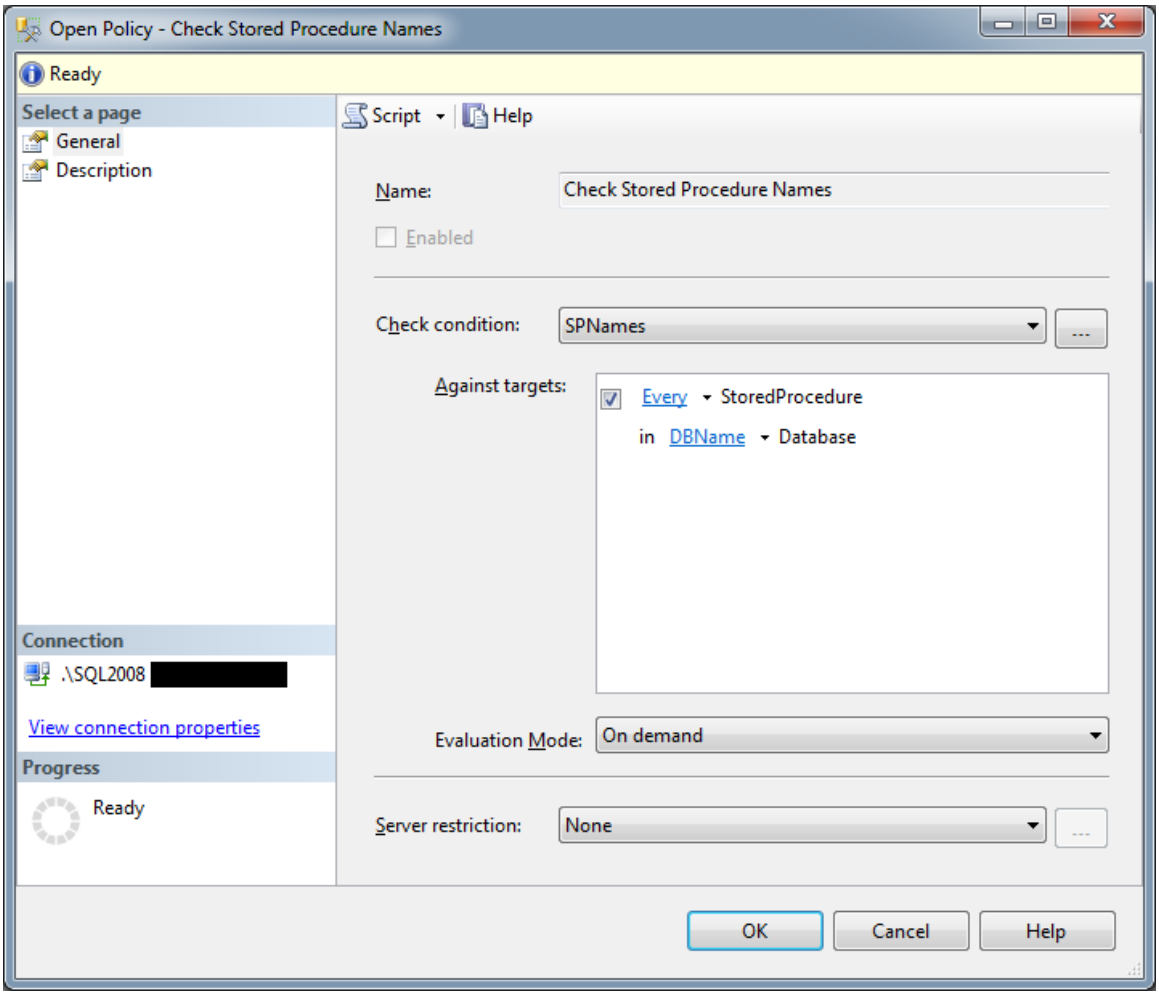


Figure 2.6: Policy-Based Management – Policy properties.

For this example, I will not add a condition for the **Server restriction** as I am only using a single SQL Server instance for this post.

We click on **OK** and the Policy is ready. As this Policy's Evaluation Mode has been set to "**On demand**" we cannot enable it. It wouldn't make any sense after all. We can only use it by right-clicking on it and selecting "**Evaluate**". This is the purpose of the **On Demand** evaluation mode.

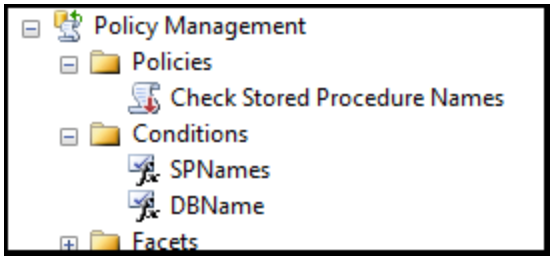


Figure 2.7: Policy-Based Management – Policies and Conditions in SSMS.

After running the Policy we get its evaluation results:

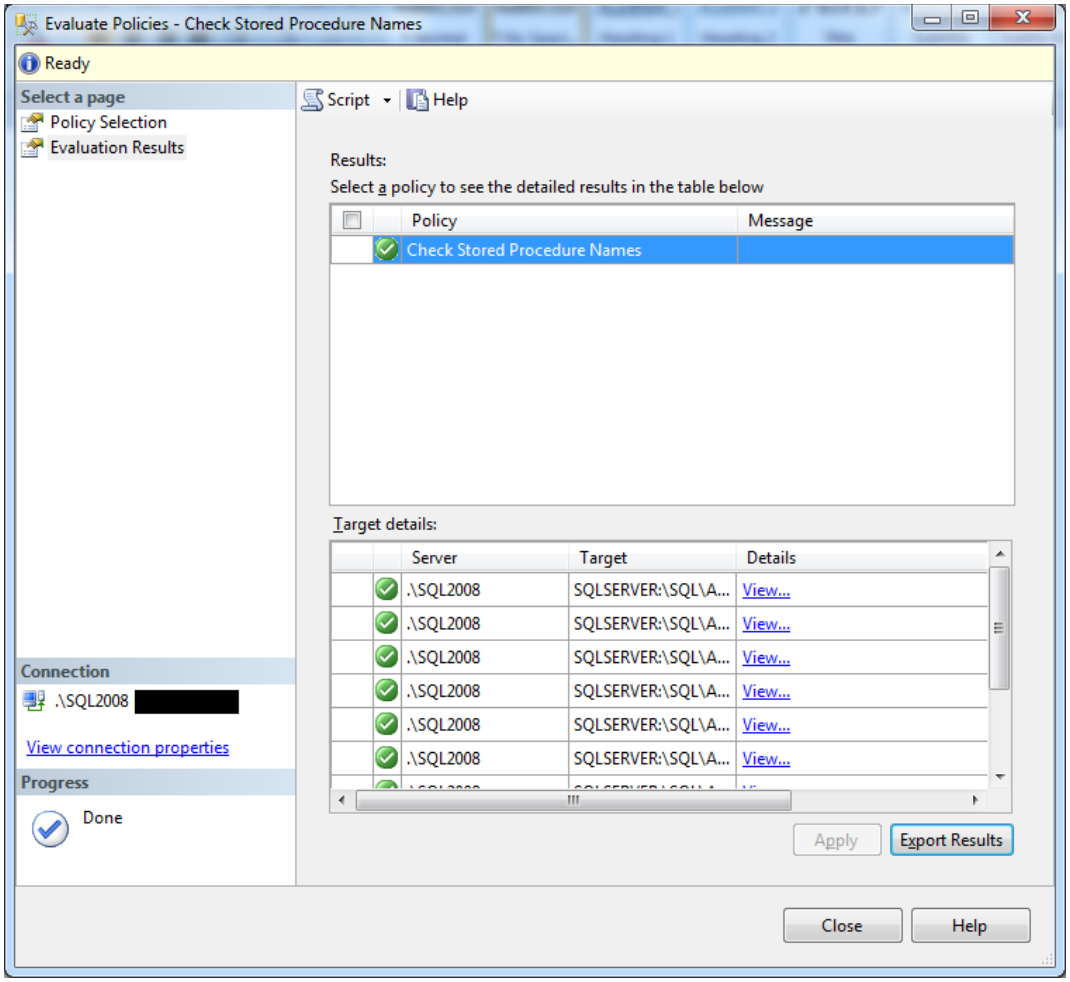


Figure 2.8: Policy-Based Management – Running a Policy.

From the above screenshot we can conclude that at the time being all stored procedures within the "AdventureWorks2008" database comply with the condition set up in the Policy.

Now, what about adding a new procedure on the "AdventureWorks2008" database having the name "**sp_SampleSP**"? Let's do so, re-run the Policy and check the evaluation results.

Here are the results:

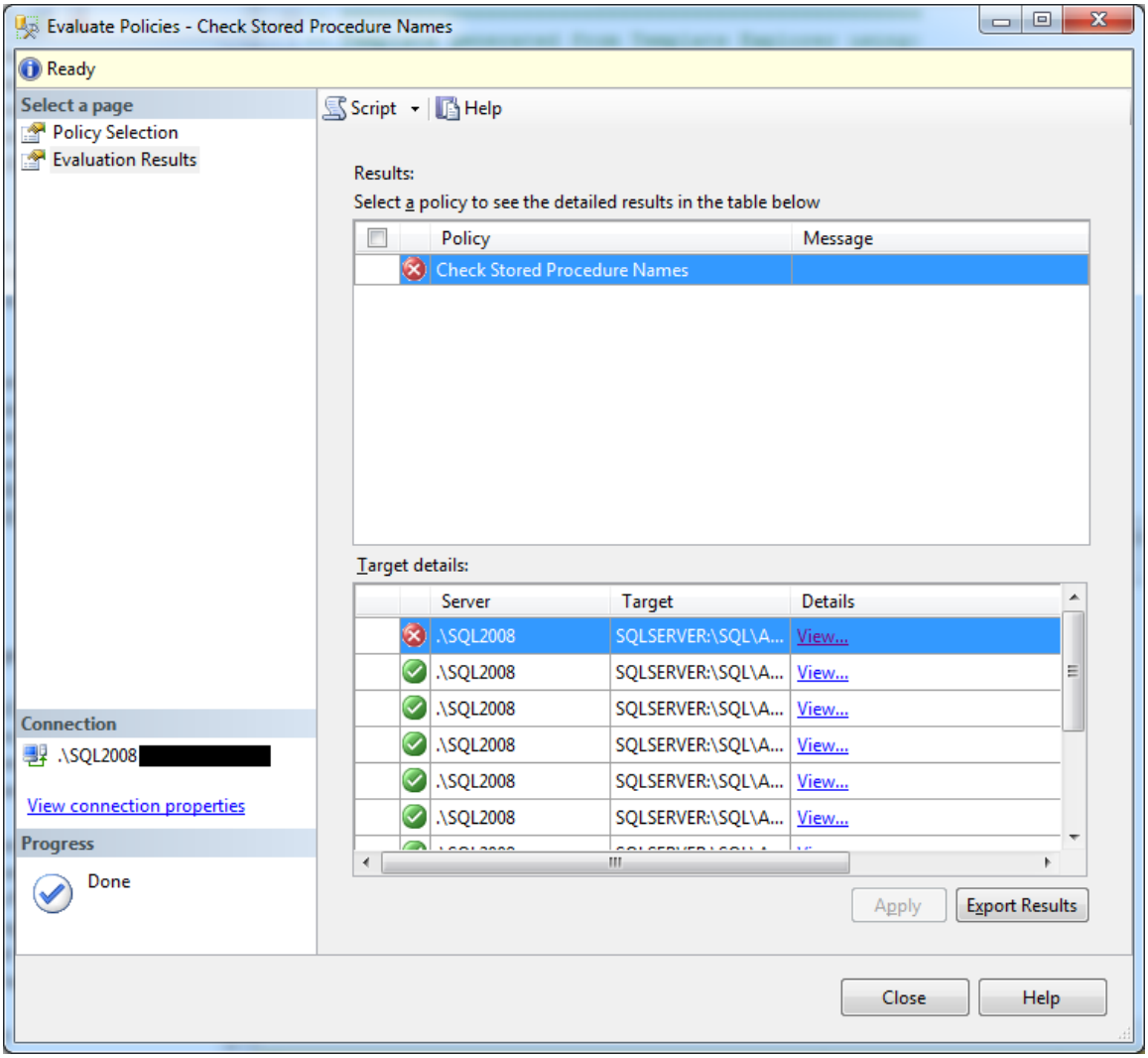


Figure 2.9: Policy-Based Management – Policy evaluation results.

As you can see, the execution of the policy returned an error. By clicking on **[Details] View…** we can see what the policy found:

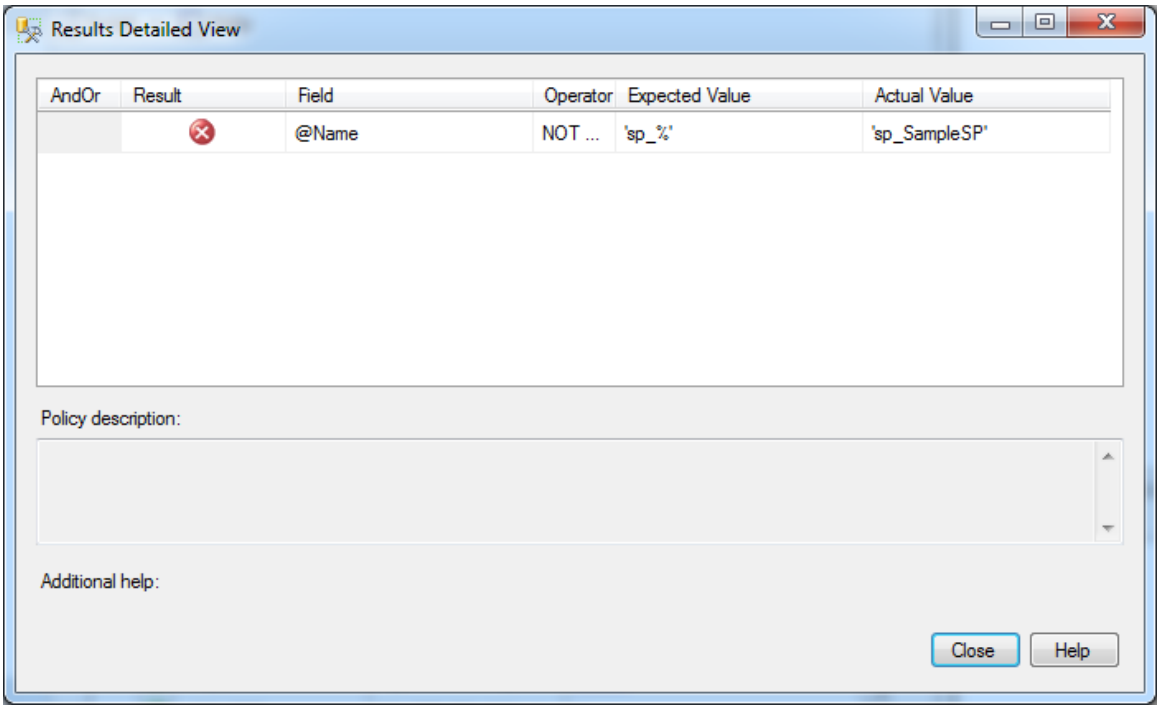


Figure 2.10: Policy-Based Management – Policy evaluation details.

We can see that the Policy reported that the stored procedure "**sp_SampleSP**" does not comply with the policy's condition of **not** having stored procedures with names starting with '**sp_**'.

The above is a very basic example of what a Policy can do. The only limit is your creativity.

On change: prevent

If we want to change the Evaluation Mode of the above Policy to "**On change: prevent**", we can double-click on the Policy, change it and then enable the Policy:

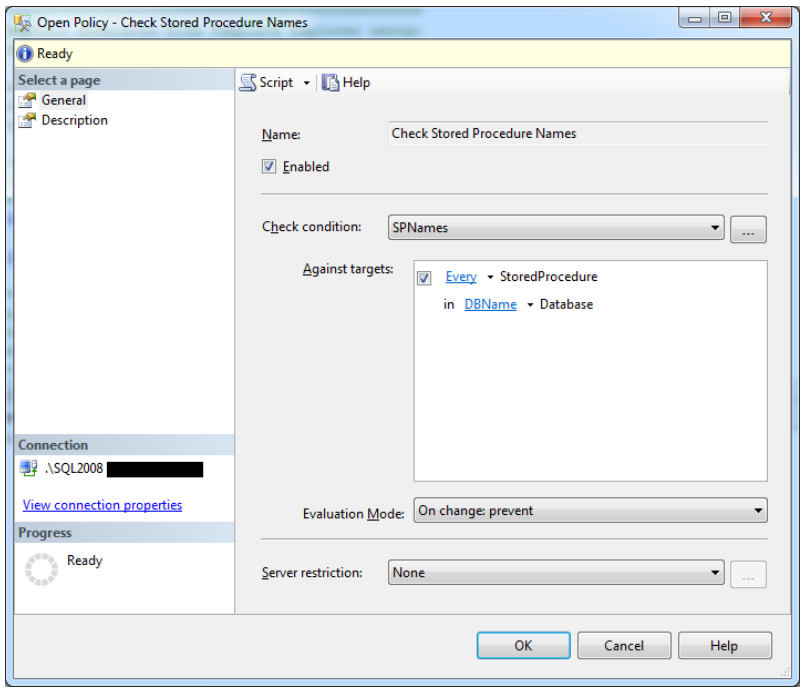


Figure 2.11: Policy-Based Management – Using Policies for preventing certain actions.

Now let's try to create again the same stored procedure:

Figure 2.12: Policy-Based Management – Policy prevention mechanism in action.

As you can see from the above screenshot, the Policy prevented the Database Engine from creating the stored procedure as it does not comply with the naming condition specified by the Policy!

On change: log only

Similarly, we can change the Evaluation Mode of the Policy to "**On change: log only**". This will allow the stored procedure to be created but it will also add a log entry in **SQL Server Logs** and **Windows Application Log** reporting the violation of the Policy:

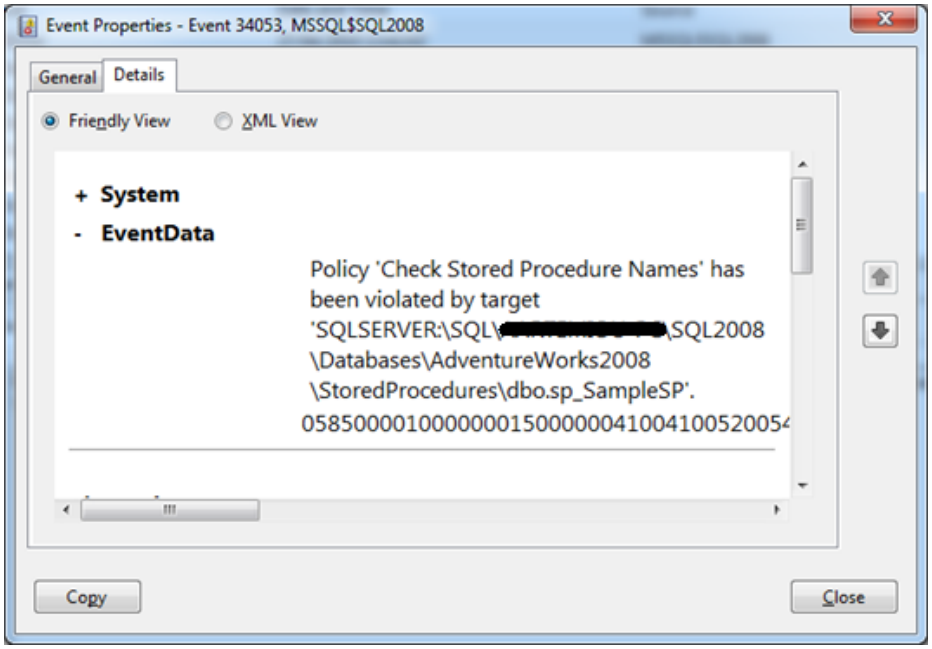


Figure 2.13: Policy-Based Management – Policy log-only action – Windows logs.

Figure 2.14: Policy-Based Management – Policy log-only action – SQL Server logs.

On schedule

Last but not least, you can set up a schedule for when the Policy should be executed. This will create a SQL Server Agent job that will execute the Policy at the selected time.

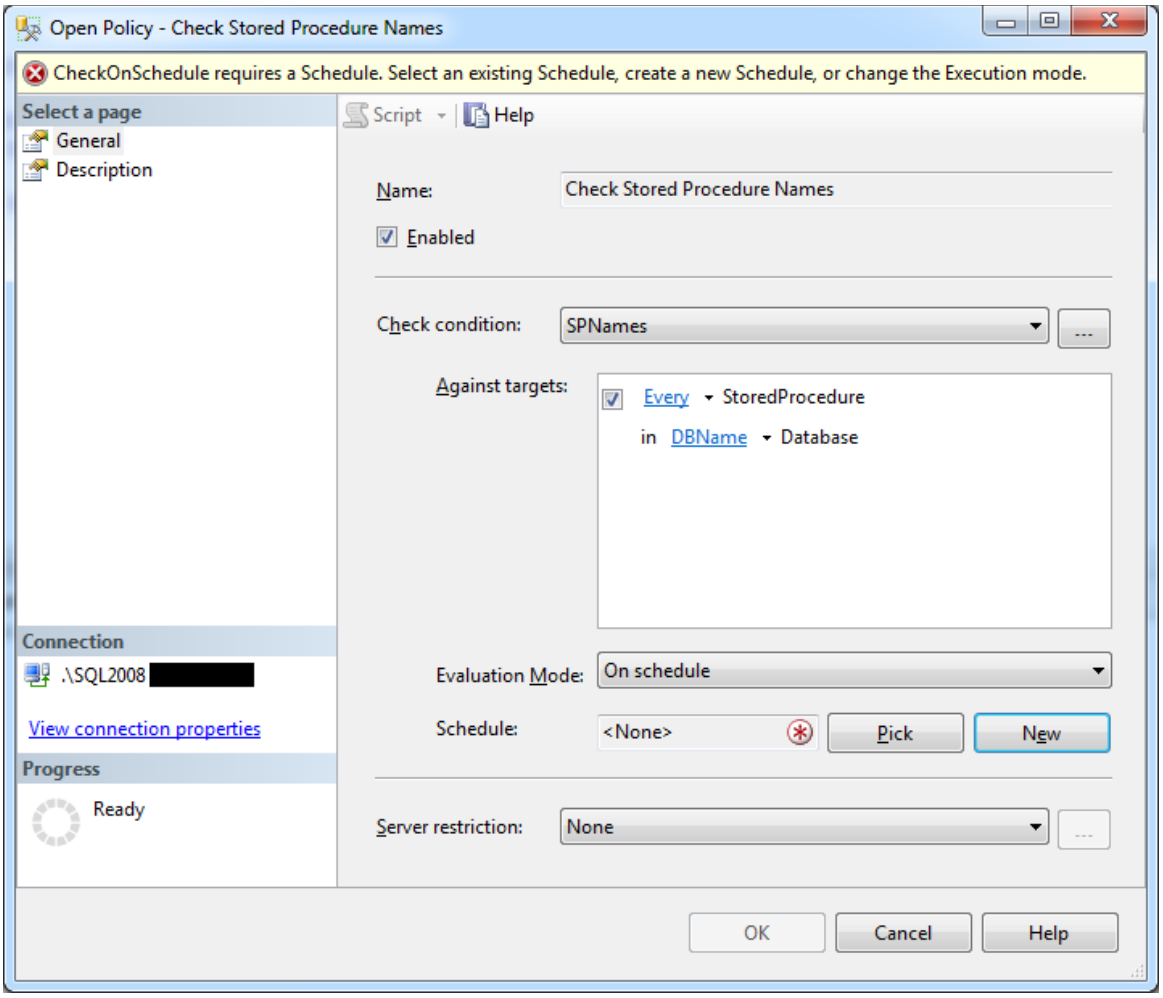


Figure 2.15: Policy-Based Management – Set schedule for Policy evaluation.

This article was a very basic example on how you can use Policy-Based Management in SQL Server 2008. You can build really advanced Policy scenarios where you can have multiple Policies and conditions, evaluating a large set of actions in more than one SQL Server instances.

The Policy-Based Management feature in SQL Server 2008 is extremely powerful and allows setting up strict Compliance scenarios ensuring the health and desired structure of the monitored SQL Server instances and the databases that exist within them.

The best way to learn more about Policy-Based Management in SQL Server is to get experimented with it in order to fully realize its potentials. You can also check out this link for more information on Policy-Based Management in SQL Server.

➔ ***Applies to: SQL Server 2008 or later.***

Migrating to a Contained Database in SQL Server 2012

(source: http://aartemiou.blogspot.com/2012/07/migrating-to-contained-database-in-sql.html)

In 2012, one of my articles on Contained Databases in SQL Server 2012 was published on the MVP Award Program Blog. In that article I explained with an example how you can easily create and access a partially contained database from scratch.

In this article I am describing how you can convert a "normal" database to a partially contained database thus making it fully portable.

A typical security/user access configuration for a database would be set up as follows:

- Create the login (SQL or Windows) under the SQL Server Instance "Security" module
- Set the proper "User Mapping" along with the corresponding database role membership(s).

This is illustrated in the following screenshot where we see the user access configuration for the sample database "**NormalDB**":

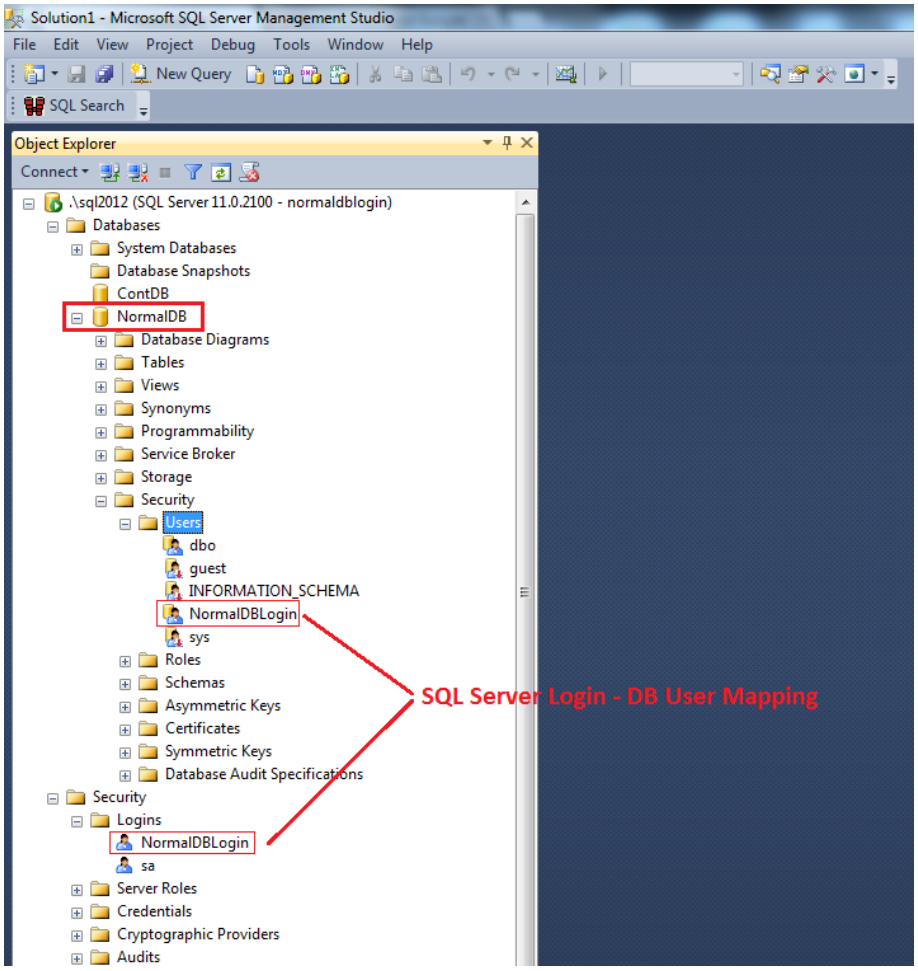


Figure 2.16: Typical user access configuration for a database.

In case you migrate the database to another SQL Server instance and you do not migrate the SQL login as well, you will encounter the issue of orphaned database users (not associated to a SQL login).

With partially contained databases there is not such an issue for the simple reason that you do not need to have a SQL login associated to the database user as you just need the database user.

So, let's see how we can convert "NormalDB" to a partially contained database.

First of all, we need to enable "**contained database authentication**" on the SQL Server instance if not already enabled:

```
USE [master];
GO

sp_configure 'contained database authentication', 1;
GO
RECONFIGURE;
GO
```

Listing 2.1: Enabling contained database authentication.

Then, we change the containment option for the database to "**PARTIAL**":

```
ALTER DATABASE [NormalDB]
SET CONTAINMENT=PARTIAL;
GO
```

Listing 2.2: Changing the containment type to partial.

And here's the last step where the magic takes place:

```
USE [NormalDB];
GO

sp_migrate_user_to_contained
@username = N'NormalDBLogin',
@rename = N'keep_name',
@disablelogin = N'disable_login';
GO
```

Listing 2.3: Migrating the database user for use with contained database authentication.

sp_migrate_user_to_contained is a special stored procedure shipped with SQL Server 2012 and its purpose is to remove dependencies between a database and the SQL Server instance that hosts it. More specifically, it separates the user from the original SQL Server login.

In the above example, what "sp_migrate_user_to_contained" did was to set the login's password to the contained database user and then disable the SQL Server login.

For accessing the contained database you can use the following steps:

Set the database to connect to (NormalDB):

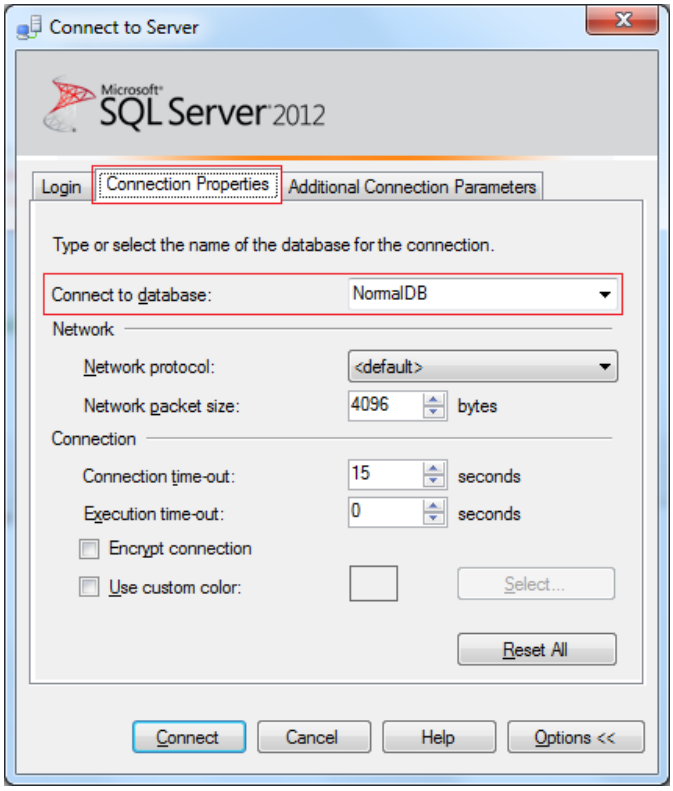


Figure 2.17: SSMS connection properties – select specific database to connect to.

Enter the contained database's user credentials:

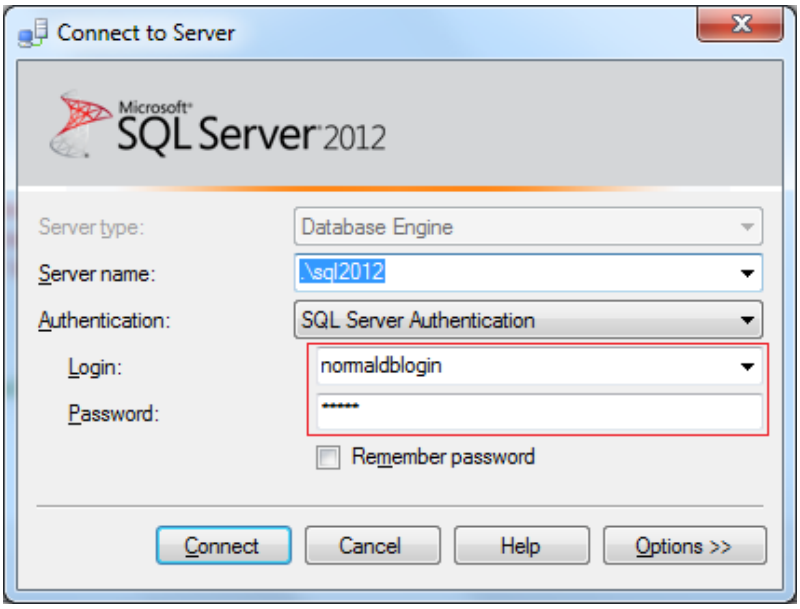


Figure 2.18: SSMS connection properties – connect with contained user.

As you can see, you have managed to connect to the contained database and access its objects:

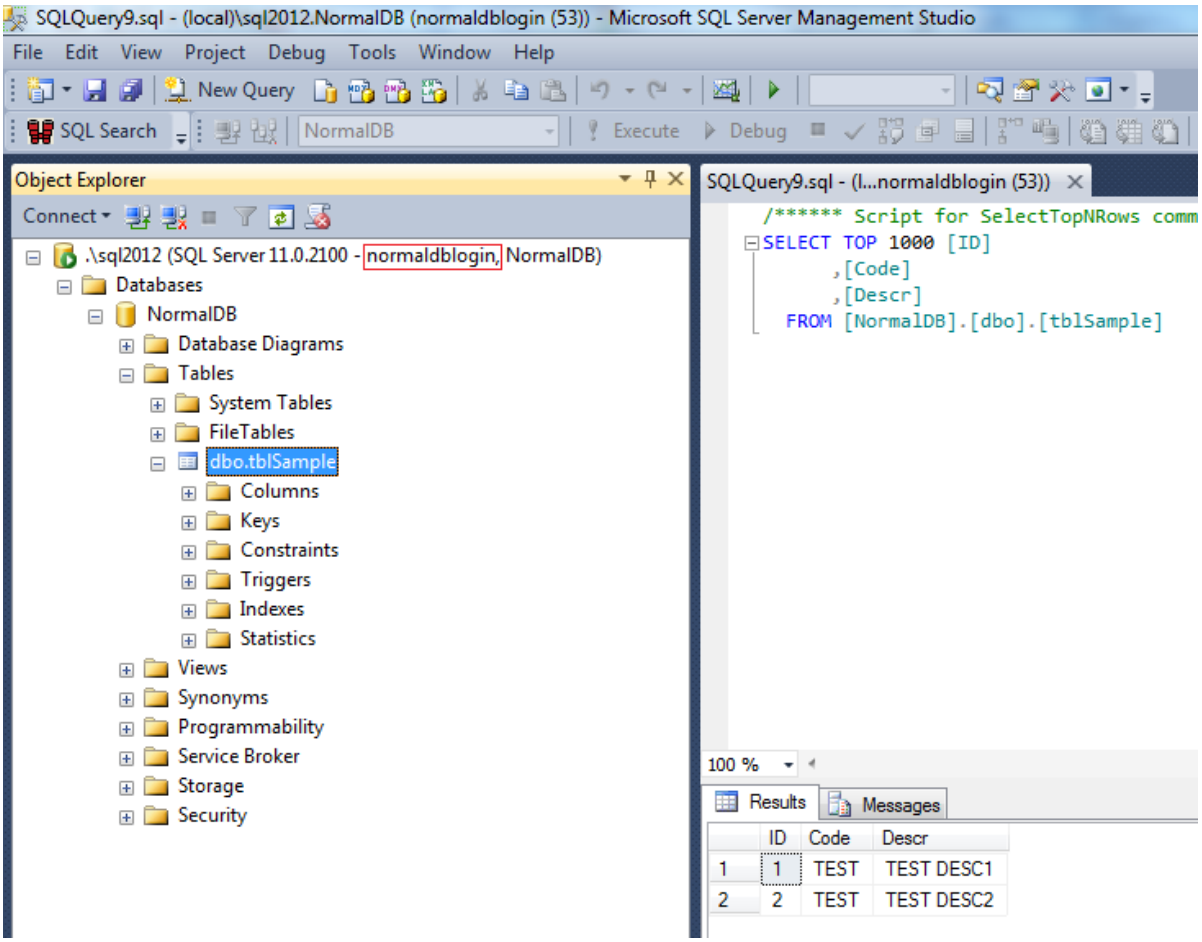


Figure 2.19: Successful connection to contained database.

➔ ***Applies to: SQL Server 2012 or later.***

▪ Changing the Database Owner in a SQL Server Database

(source: http://aartemiou.blogspot.com/2010/08/changing-database-owner-in-sql-server.html)

In SQL Server, the **dbo** (Database Owner) is a special user that has all the necessary permissions to perform all activities in the database. This user exists in all SQL Server databases.

Usually, there are SQL Server or Windows logins mapped to the dbo user, thus "inheriting" the permissions of the database's dbo user and being able to perform all the activities in the database that the dbo user can perform.

In some cases, due to various changes a DBA might perform in SQL Server, you may discover that the dbo user is orphaned, meaning that there is not any login mapped to this user.

If you would like to change this in SQL Server 2000, thus mapping a login to the dbo user you can make use of the "**sp_changedbowner**" system stored procedure.

The syntax as provided in the T-SQL reference library is the following:

```
sp_changedbowner [ @loginame= ] 'login'
[ , [ @map= ] remap_alias_flag ]
```

Listing 2.4: Changing the database owner in SQL Server 2000.

For example, if you want to make the SQL Server Login '**Tom**' the owner of the database "**TestDB**" you can use the following T-SQL script:

```
USE TestDB;
GO
EXEC sp_changedbowner 'Tom';
GO
```

Listing 2.5: Example of changing the database owner in SQL Server 2000.

Additionally, if you want to make a Windows Login, for example '**SampleDomain\TestUser**' the owner of the database "**TestDB**" you can use the following T-SQL script:

```
USE TestDB;
GO
EXEC sp_changedbowner 'SampleDomain\TestUser';
GO
```

Listing 2.6: Setting a Windows login as the database owner for a database.

Please also note that only members of the **sysadmin** fixed server role can execute the stored procedure sp_changedbowner.

In SQL Server 2005 or later, you can make use of the "ALTER AUTHORIZATION" command as sp_changedbowner is deprecated and will be removed in a future version of SQL Server.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Creating Logins for Orphaned SQL Server Users

(source: http://aartemiou.blogspot.com/2010/02/creating-logins-for-orphaned-sql-server.html)

There are cases where you might need to restore an entire SQL Server database (i.e. in the case of a corrupted database that cannot be recovered, etc.). A set of objects that are restored from the backup set and are included in the restored database are the **database users**.

When the database is restored, these users might be orphaned as the corresponding SQL Server logins might not exist anymore (especially when you restore the database on another instance).

In this case, you can use the following stored procedure for fixing such issues:
sp_change_users_login

Example

```
USE AdventureWorks;
GO

EXEC sp_change_users_login 'Auto_Fix', 'UserName', NULL, 'Password';
GO
```

Listing 2.7: Fixing the orphaned database user issue.

The above example will create a new SQL Server Login for the given user name and use as a password the given password string. You can also assign existing logins to orphaned users.

For more information you can visit SQL Server Books Online.

➔ ***Applies to: SQL Server 2000 or later.***

▪ Transparent Data Encryption in SQL Server 2008

(source: http://aartemiou.blogspot.com/2008/12/transparent-data-encryption-tde-in-sql.html)

Transparent Data Encryption (TDE) is a data security feature that first shipped with SQL Server 2008. TDE performs real-time I/O encryption and decryption of the data and log files, that is the entire database. For achieving that, it uses a database encryption key stored in the database boot record.

A derived benefit of TDE is that whenever a database using TDE is backed up, the backup set is also encrypted. All of the above provide significant data security in SQL Server 2008 and later versions. The procedure for encrypting a database is provided below by a T-SQL script example:

```
USE master;
GO

--Step 1: Create a Master Key
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password_goes_here';
GO

--Step 2: Create or obtain a certificate protected by the master key
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'MyCertificate';
GO

--Step 3: Create a database encryption key and protect it by the certificate
USE [DATABASE_NAME]
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO

--Step 4: Set the database to use encryption
ALTER DATABASE [DATABASE_NAME]
SET ENCRYPTION ON;
GO
```

Listing 2.8: Encrypting a database using Transparent Data Encryption.

After the above are performed, the database will enter the "Encrypted" state.

Remarks:

1. Important note from Books Online: When enabling TDE, you should immediately back up the certificate and the private key associated with the certificate . This is absolutely necessary when trying to restore or attach the encrypted database on another server because you will need to use these keys and certificates. In the opposite case the database will not be accessible. Additionally the encrypting certificate should be retained even if TDE is no longer enabled on the database as it may need to be accessed for some operations.

2. Steps 3 and 4 can be performed from within SQL Server 2008 Management Studio by right-clicking on the database and selecting **Tasks → Manage Database Encryption**.

3. Note that in SQL Server 2008, four encryption algorithms are provided:

- AES_128
- AES_192
- AES_256
- Triple_DES

4. The entire TDE on a database is completely transparent to the user as it is performed in the background and on the fly.

More information regarding Transparent Data Encryption in SQL Server 2008 can be found in the following link.

➔ ***Applies to: SQL Server 2008 or later.***

▪ Encrypting SQL Server Databases

(source: http://aartemiou.blogspot.com/2013/02/encrypting-sql-server-databases.html)

In corporate environments there is always the requirement/policy of data protection. Data is the most valuable asset in every organization, after its human resources of course, as it is with data where the organization's business processes run and produce results.

SQL Server is one of the leading data platforms worldwide and as such, in SQL Server 2008 (Enterprise Edition) and later, the data protection mechanisms have been significantly enriched.

SQL Server 2008 introduced Transparent Data Encryption (TDE); a mechanism allowing the DBA to easily encrypt databases without affecting their operation as it allows full transparency to the database users and applications. Based on TDE, the DBA encrypts the database using a master key and a certificate. The user accesses the database transparently as the encryption is automatically maintained on the Database Engine-level and in the case of a potential data theft, i.e. someone illegally copies the database files (data and log files), when he tries to access the database it will not be allowed because the database is encrypted. You can find a simple example on how you can use Transparent Data Encryption in the previous article of this chapter.

As mentioned above, Transparent Data Encryption is available in SQL Server 2008 or later. However, if you would like to achieve database encryption in earlier versions of SQL Server, you can do it by using a special feature of Windows on the file system-level, that is Encrypting File System (EFS) which was introduced in version 3.0 of NTFS and provides file system-level encryption. EFS is available from Windows 2000 onwards. The way EFS works is transparent, yet very powerful. A high-level description of its operation is the following.

For encrypting files:

1. With a symmetric file encryption key (FEK) generated for this purpose, EFS encrypts the file.
2. Then it encrypts the FEK using the active Windows user's public key.

For decrypting files:

1. EFS decrypts the encrypted FEK using the Windows user's private key.
2. Then it decrypts the encrypted file using the decrypted FEK.

So, in the case you would like to encrypt a database in SQL Server 2005 or earlier using EFS you could achieve it by following the steps below:

1. Log in to Windows using the user account that is used by SQL Server Database Engine.
2. Log in to SQL Server Management Studio (SSMS).
3. Detach the database from the SQL Server Instance.

4. Encrypt the database files (data and log files) on the file system-level ([File Properties] - [Advanced] - [Encrypt contents to secure data]).
5. Verify the encryption of the database files using the Windows command "cipher.exe" in the folder that contains the database files. It will return the value "E" for the encrypted files.
6. [**CRITICAL**] Export the Windows user's personal certificate with the private key ([Start] - [Run] – "certmgr.msc") and store it on a safe location (i.e. removable storage).
7. Attach the database back to the SQL Server Instance.
8. If any problems, try to restart the SQL Server Instance.

Note: Be extremely cautious when running the above procedure because in the case you change the user account that runs the SQL Server Instance and the new user is not authorized to access the database files, then you will lose access to your data. Always backup the user's certificate/private key which can be used to decrypt the encrypted files. Always be careful when using data encryption mechanisms.

Note: If you encrypt database files and attempt to start the SQL Server service with a user other than the user that encrypted the database data/log files in the first place, the instance will not start and/or your database will be set to suspect.

A last thing you should have in mind when using EFS for encrypting SQL Server 2005 or earlier databases, is that EFS will affect the performance of SQL Server as there is an overhead whenever decrypting the underlying database files. There are many parameters that can be taken into consideration when it comes to the SQL Server performance degradation such as: the supporting storage system and its RAID levels, the OS configuration and more.

Related Microsoft Articles:

- Transparent Data Encryption (TDE)
- How to Export a Certificate
- Encrypting File System on TechNet Library
- Export a certificate with the private key

➔ ***Applies to: SQL Server 2000 or later.***

▪ Retrieving Security-Related Info for SQL Server Logins

(source: http://aartemiou.blogspot.com/2013/01/t-sql-tip-retrieving-security-related.html)

SQL Server Database Engine provides a variety of built-in functions that can be used for retrieving useful information about different objects in your SQL Server instance like databases, users, logins, etc.

For example, when you want to retrieve security-related information about all the logins in your SQL Server instance you can use the following T-SQL script:

```
--
-- Retrieves Security-Related Information
-- for all the SQL Server Logins
--
-- SQL Server versions  supported: SQL Server 2005 or later
--
SELECT
[name] as LoginName,
LOGINPROPERTY ([name] , 'DefaultDatabase') as DefaultDatabase,
LOGINPROPERTY ([name] , 'DaysUntilExpiration') as DaysUntilExpiration,
(CASE ISNULL(LOGINPROPERTY ([name] , 'IsExpired'),0) WHEN 0 THEN 'False' ELSE
'True' END) as IsExpired,
(CASE ISNULL(LOGINPROPERTY ([name] , 'IsLocked'),0) WHEN 0 THEN 'False' ELSE
'True' END) as IsLocked,
LOGINPROPERTY ([name] , 'PasswordLastSetTime') as PasswordLastSetTime,
LOGINPROPERTY ([name] , 'PasswordHashAlgorithm') as PasswordHashAlgorithm
FROM master..syslogins;
GO
```

Listing 2.9: Retrieving security-related information for SQL logins.

For more info, check out the following links:

- sys.syslogins
- LOGINPROPERTY

➔ ***Applies to: SQL Server 2005 or later.***

▪ Security Changes in SQL Server 2008

(source: http://aartemiou.blogspot.com/2010/09/security-changes-in-sql-server-2008.html)

SQL Server 2008, among other, introduced significant security changes that enhance the database administrators applying an even stricter security policy on the SQL Server instances.

For example a major change was the discontinuation of the Surface Area configuration tool and the introduction of the powerful Policy-Based Management feature. Another change was the Kerberos support for named pipes and shared memory protocols. However, the change that has the strongest effect on the way that many of us used to work, is that the local Windows Group **BUILTIN\Administrator** is no longer by default included in the SQL Server sysadmin fixed server role on new SQL Server 2008/R2 and later installations.

So, what does this mean? It means that if you try to access a SQL Server 2008 (or later) instance using a local administrator user account without explicitly granting him the sysadmin server role on the instance, you will not be able to have administrative rights on the instance. Actually, if this user has not any permission on the specific instance, he will not be able to access the instance at all.

Someone might say that this makes things more complicated but the truth is that it does not. It is an excellent security enhancement that actually separates Windows administrator accounts from SQL Server administrators as in large organizations these roles are different.

Additionally you need to be very careful when you install and perform the initial setup of a SQL Server 2008 (or later) instance because if you don't include at least one user in the sysadmin role, you will be locked out of that instance.

For more information on the security changes in SQL Server 2008 and SQL Server 2008 R2 you can visit this MSDN Library article. Make sure that you read it before configuring the security of your new SQL Server instance.

➔ ***Applies to: SQL Server 2008 or later.***

▪ The SELECT ALL USER SECURABLES Permission in SQL Server 2014

(source: http://aartemiou.blogspot.com/2013/07/the-select-all-user-securables.html)

The SELECT ALL USER SECURABLES permission in SQL Server 2014 is very useful new server-level permission.

When a login is granted this permission, it can view the data in all databases that the user can connect to.

For example, consider a scenario where you have the following three databases:

- db1
- db2
- db3

Each database has a different login that accesses its data. So if you do not have "sysadmin" access on the SQL Server instance, in order to access the data in all databases you will have to log in three times, each time using a different login.

However, if the DBA creates a new login (for example **db123**) and grants it with the "SELECT ALL USER SECURABLES" permission, then the new login will be able to access the data in all three databases.

Using SSMS GUI, you can achieve the above using two steps:

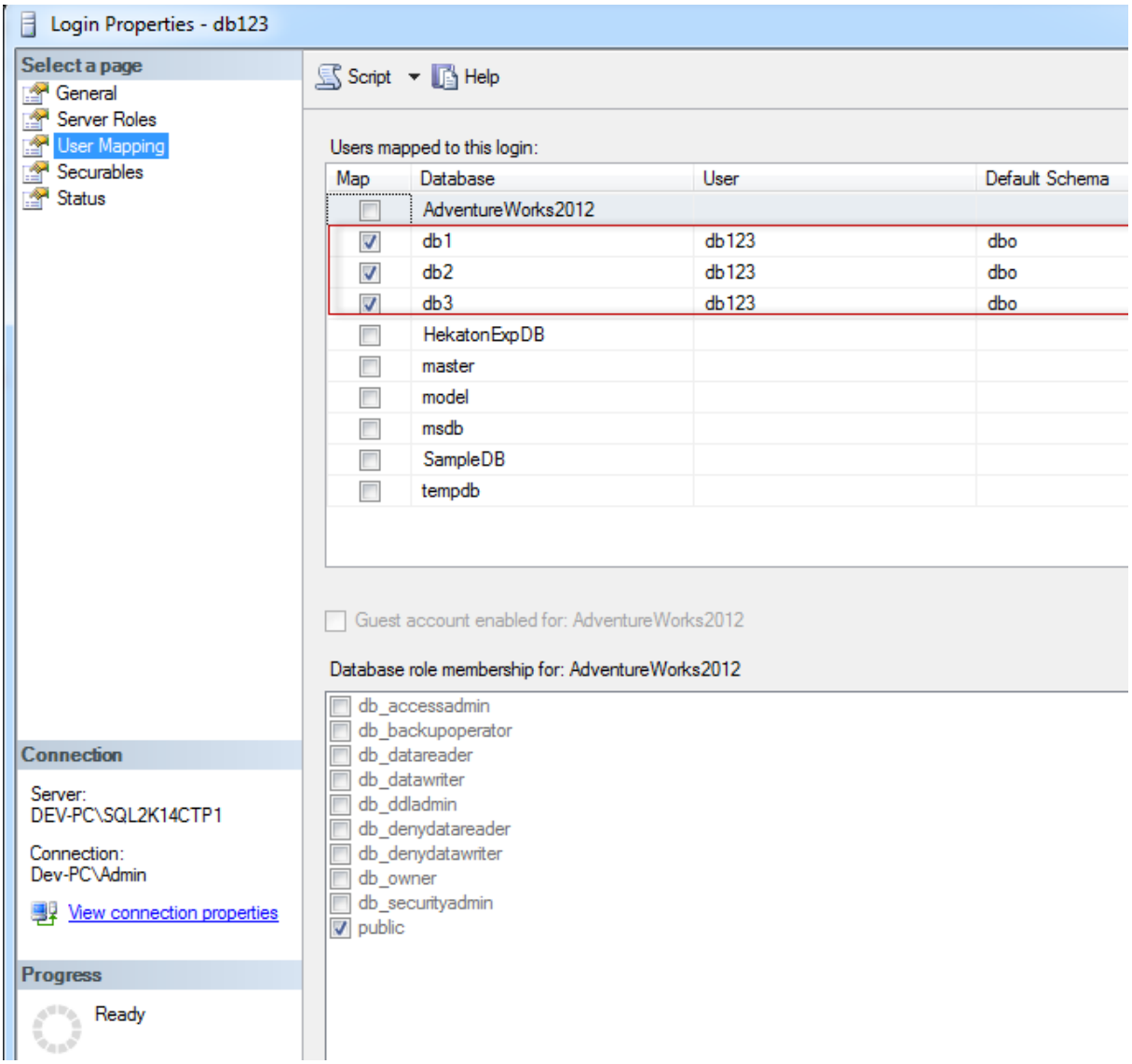


Figure 2.20: Map the login (db123) to the default schema of DB1, DB2 and DB3.

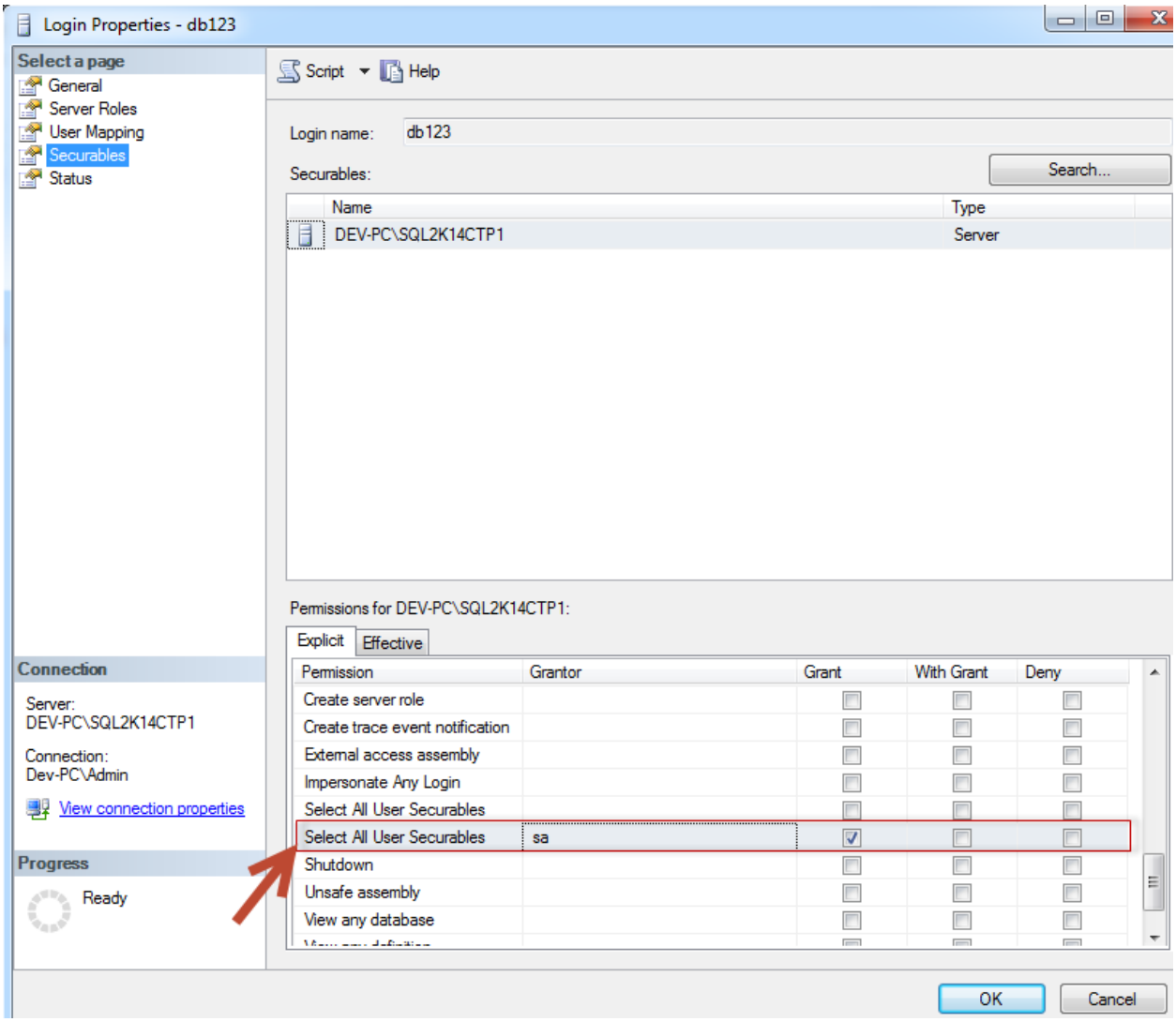


Figure 2.21: Grant the login (db123) with the "Select All User Securables" permission.

As you can see in the last screenshot, it was possible to retrieve the data from all three databases by just using the login **db123** which was only granted with the server-level permission "Select All User Securables":

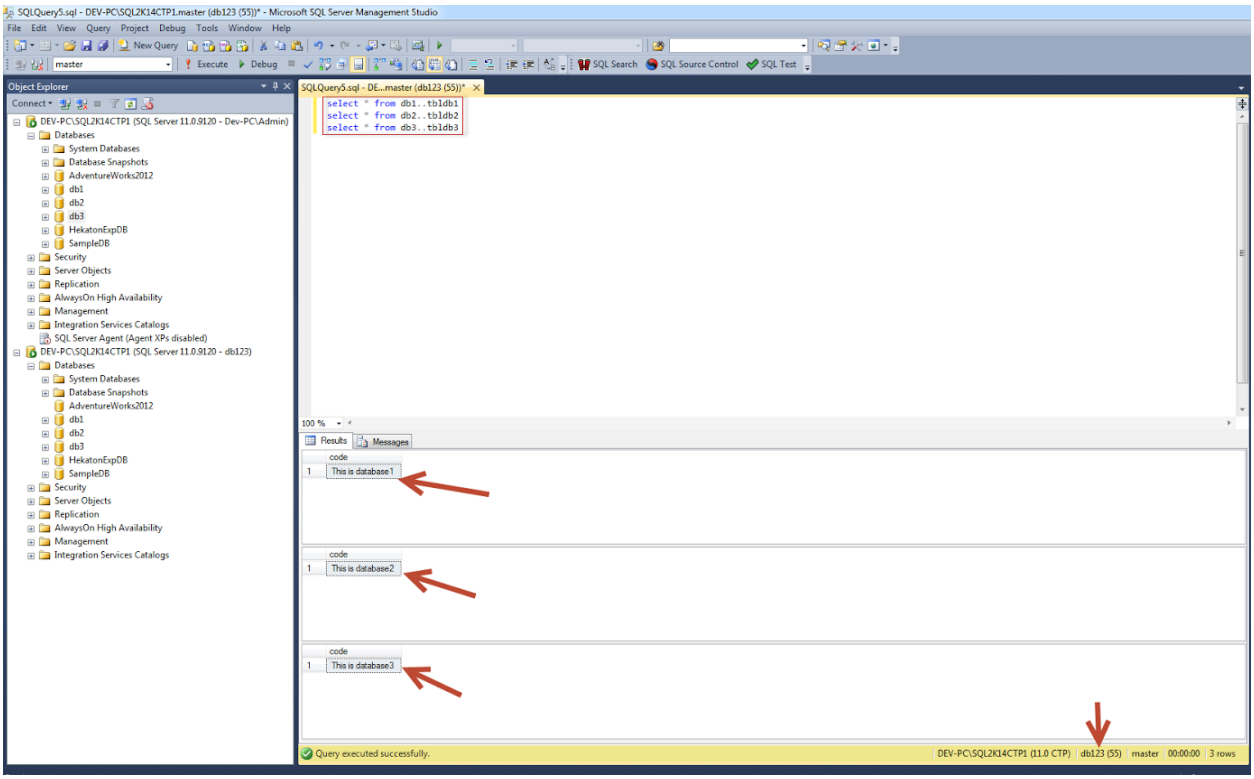


Figure 2.22: Successful database access with the "Select All User Securables" permission.

Summary

In this chapter you learned more about SQL Server's security and data governance features. Among other you learned how to enforce data policies with Policy-Based Management and how to protect your data from unauthorized access with the use of different encryption methods. Last but not least you learned how to manage database logins as well how to migrate to contained databases in SQL Server 2012. The next chapter talks about selected topics having to do with integrating a SQL Server instance with other instances as well as accessing other resources within the network.

➔ ***Applies to: SQL Server 2014 or later.***

CHAPTER 3

Integration

IN THIS CHAPTER:

- How to Create a Simple Linked Server Between SQL Server Instances
- Using Proxies in SQL Server Agent Jobs
- Using Unicode in SQL Server

Integration can be defined as the set of practices and procedures that are used for building large systems that consist of other smaller systems that can be called subsystems.

It is not random the fact that the majority of large software companies in the market, provide Application Programming Interfaces (APIs) thus allowing their systems to integrate with other third-party systems. Furthermore, along with these APIs, you can also find in the market ready-made API implementations for many of these systems. These implementations can be found under different flavors such as: middleware, web services, source code libraries, etc.

In the RDBMS world you can often find these API implementations as *Data Providers*. To this end, if you are using *RDBMS A* you can interconnect it with *RDBMS B* via a provider.

Some examples of such integrations include: creating a linked server in SQL Server targeting another RDBMS in order to access the data from within SQL Server, retrieving data from different RDBMSs in PowerPivot, building an SSIS package that retrieves and transforms data from several data sources and so on.

SQL Server provides all the tools that make it easy to integrate SQL Server instances to other instances or even other DBMSs and generally interact with other systems in the network.

This chapter shows how to create linked servers between different SQL Server instances, how to enable SQL Server Agent jobs to interact with other systems and network resources using proxies and last, how a database can support multiple alphabets by using Unicode data types.

How to Create a Simple Linked Server Between SQL Server Instances

(source: http://aartemiou.blogspot.com/2012/09/t-sql-tip-how-to-create-simple-linked.html)

There is many times where you need to "connect" your SQL Server instance with another one in order to run queries and data development work on both instances from the same environment (i.e. within the same SSMS query window). It is easy to do this by using Linked Servers.

The following example shows how you can create a simple linked server between two SQL Server instances:

```
USE [master]
GO

--Create the linked server object
EXEC master.dbo.sp_addlinkedserver
@server = N'[LinkedServerReferenceName]',
@srvproduct=N'',
@provider=N'SQLNCLI',
@datasrc=N'[DestinationSQLInstanceName]';

--Set up the user mapping between local and remote instances
--You can run the process below for as many local logins you
--want to have access to the destination SQL Server instance
EXEC master.dbo.sp_addlinkedsrvlogin
@rmtsrvname=N'[LinkedServerName]',
@useself=N'False',@locallogin=N'[LocalLogin_SQL_or_Windows]',
@rmtuser=N'[Remote_SQL_Login_Name]',
@rmtpassword='[Remote_SQL_Login_Password]';
GO

--Example of querying a remote table
SELECT * FROM [LinkedServerName].[Database_Name].[Schema_Name].[Table_Name];
GO
```

Listing 3.1: Creating a linked server between two SQL Server instances.

For more info, check out the following links:

- sp_addlinkedserver
- Linked Servers Theory

➔ ***Applies to: SQL Server 2000 or later.***

Using Proxies in SQL Server Agent Jobs

(source: http://aartemiou.blogspot.com/2011/08/using-proxy-accounts-in-sql-server.html)

When using SQL Server, in many cases you might need to set up a SQL Server Agent job that will be accessing a resource that is not local but exists within the domain.

For example, you want to include a step in a SQL Server Agent job that based on some logic, will be handling a Windows service on a server within the domain by using the Operating System (CmdExec) SQL Server subsystem.

In order for the job to be successfully executed, the specific job's execution context should be allowed access to the target resource in the domain.

If you are using a domain user as the service account for the SQL Server Agent in the specific instance, you can assign the necessary access rights to that user account and this will allow the Agent job step to succeed.

However, there is also another way which I personally prefer that it ***using a Proxy Account for executing the specific job step.***

In order to be able to do this you must perform the following actions within the instance of SQL Server:

1. Create a credential
2. Create a Proxy Account that uses the credential you created in the first step

For creating a credential in SSMS you navigate to: Security → Credentials
You can then create the credential by providing an identity (i.e. a domain user) along with its password.

For creating a Proxy Account in SSMS you navigate to: SQL Server Agent → Proxies
You can then create a new Proxy Account by giving it a name and performing the following:

- Provide the credential you earlier created
- Enter a description (optional)
- Set the subsystems for which the Proxy Account will be active. These are:
    - ActiveX Script
    - Operating system (CmdExec)
    - Replication Distributor
    - Replication Merge
    - Replication Queue Reader
    - Replication Snapshot
    - Replication Transaction-Log Reader
    - SQL Server Analysis Services Command
    - SQL Server Analysis Services Query

- SQL Server Integration Services Package

You can now proceed and set up the SQL Server Agent job along with its steps, and in the step you want to use the Proxy Account you select it in the "**Run as**" drop down box.

Whenever the specific job step runs, it will be executed in the context of the provided Proxy Account.

* Note that in order to be able to use a Proxy Account in a specific job step, the Proxy Account needs to be activated for the specific subsystem (i.e. Operating system - CmdExec).

➔ ***Applies to: SQL Server 2005 or later.***

▪ Using Unicode in SQL Server

(source: http://aartemiou.blogspot.com/2011/05/using-unicode-in-sql-server.html)

Unicode is the standard used in the computing industry for encoding and representing any text in the most written languages.

SQL Server supports Unicode, thus allowing the easy storage and manipulation of data in the most languages. If you want a database to support a specific alphabet, then you can make use of the database collation feature upon the database's creation process. However, if you want to enable your database to support multiple alphabets, then you can make use of SQL Server's Unicode support by using the Unicode data type nvarchar for all character data (you could also use nchar and ntext but they are deprecated and will be removed in a future version of SQL Server).

In order to test Unicode support, let's run a simple example with the Cyrillic alphabet. First, let's create two tables in SQL Server, the first one does not support Unicode and the other does:

```
CREATE TABLE tStandard(
[name] varchar(100)
);
GO

CREATE TABLE tUnicode(
[name] nvarchar(100)
);
GO
```

Listing 3.2: Creating a table with a Unicode data column.

As you can see, in the **tStandard** table the "**name**" column's data type is **varchar** and in the **tUnicode** table the "**name**" column's data type is **nvarchar**.

The difference between the varchar and nvarchar data types is that the former uses 1 byte for representing characters where the later uses 2 bytes thus supporting Unicode.

The next step is to create a text file in Unicode containing three records with Russian text:

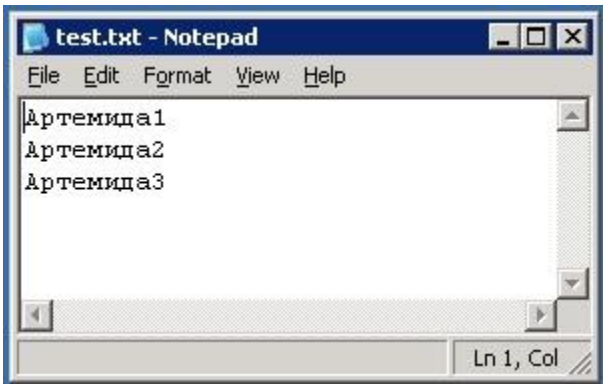


Figure 3.1: Unicode text.

The next step is to import the data into the two tables using the SQL Server Import and Export Wizard. As you can see from the screenshot below, when trying to import the data into the **tStandard** table the process failed because of the fact that the data did not match the target code page (**GREEK_CI_AS**):

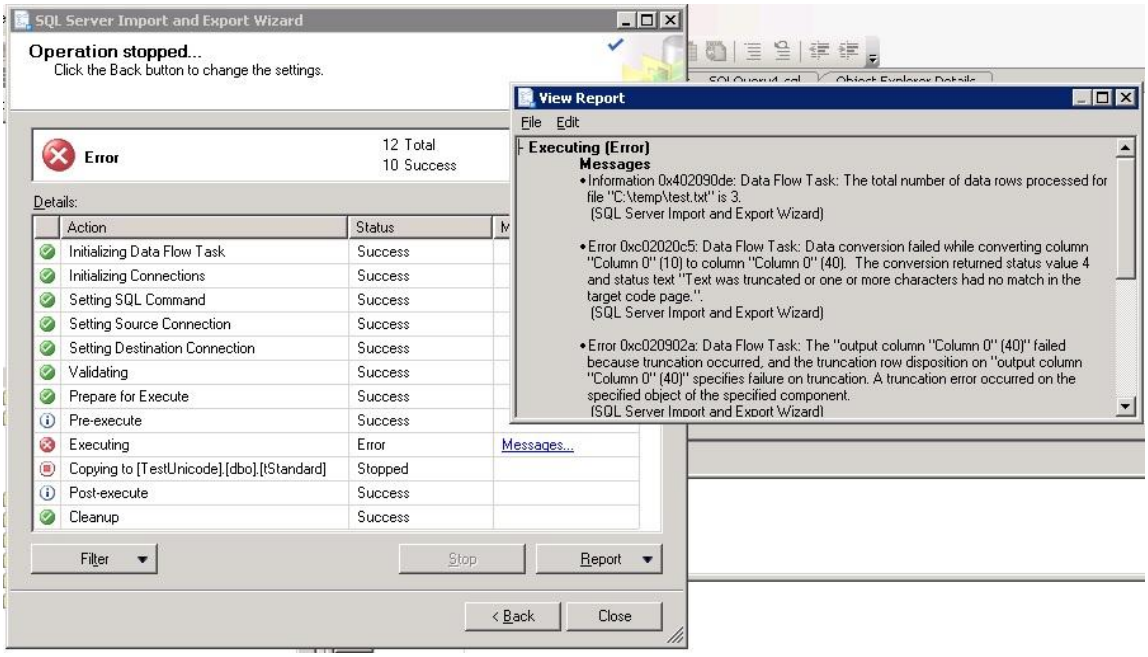


Figure 3.2: Trying to import data into a column where code pages differ.

In the above process, if the text was in Greek and not in Russian, by the time the database's collation is set to GREEK_CI_AS, the process would be successful. However, in the above example, this is not the case.

Let's see what happens when trying to import the data into the **tUnicode** table:

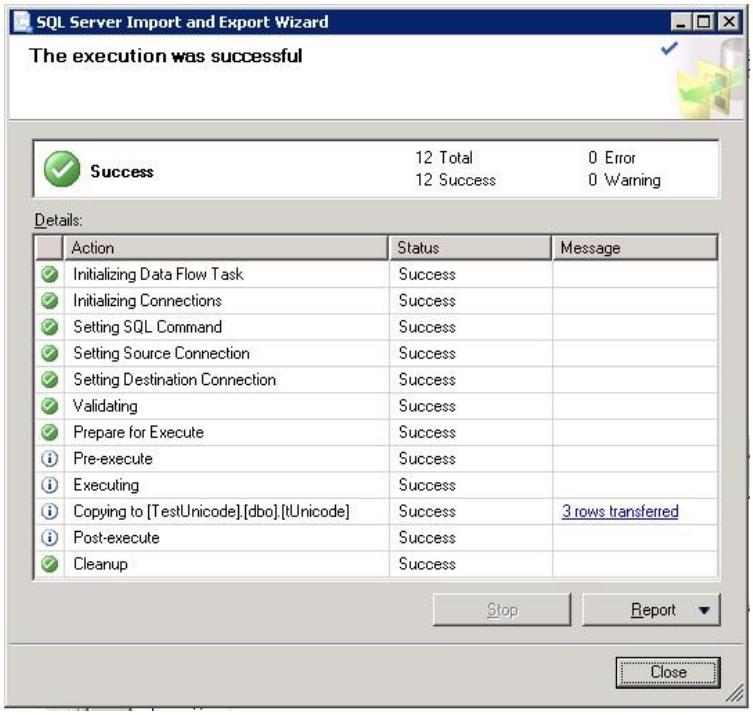


Figure 3.3: Importing data into a Unicode data column.

As you can see, the data was imported successfully.

Let's run a "select" query on both tables just to confirm the outcome of the above two processes:

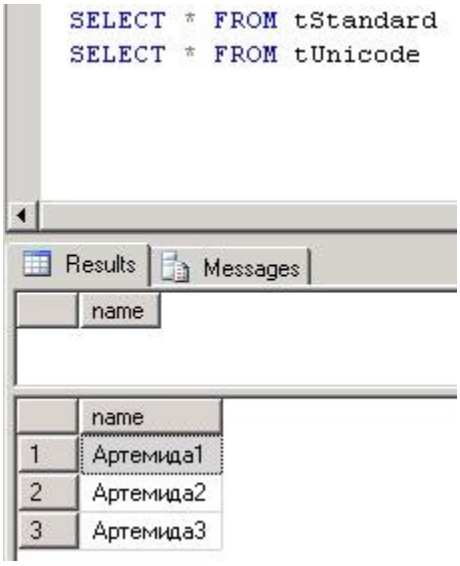


Figure 3.4: Querying a Unicode data column.

The above experiment, even simple, illustrates how you can make use of Unicode in SQL Server, in cases where the use of the database collation does not match with the alphabet of character data that you may want to process. By using the **nvarchar** data type, you can store character data in most of the written languages.

From the performance and storage aspect, the only thing you need to have in mind is that when using Unicode data types such as the nvarchar, the required storage will be doubled in comparison with using non-Unicode data types (i.e. varchar).

➔ ***Applies to: SQL Server 2000 (partially), 2005 or later (fully).***

▪ Summary

In this chapter you learned how to easily interconnect two or more SQL Server instances using linked servers. You also learned how to execute jobs in SQL Server Agent using proxies and how to design a database/table(s) that support multiple alphabets with the use of Unicode data types. The next chapter discusses different special topics such as: the Windows Internal Database (SSEE), how to dynamically generate T-SQL statements, different T-SQL tips and much more.

CHAPTER 4

Special Topics

IN THIS CHAPTER:

- Windows Internal Database (SSEE)
- Dynamically Generating T-SQL Statements
- Massively Detaching and Re-attaching Databases in SQL Server
- Installing 32-bit SQL Server 2005 Reporting Services on a 64-bit machine/Windows OS
- Useful SQL Server Knowledge

Have you ever wondered what Windows Internal Database is, what does exactly do and how you can connect to it? Have you ever wondered how you can massively manage databases and other SQL Server objects with just a single T-SQL script?

This chapter is dedicated to different special topics having to do with various administration-related tasks in SQL Server. Among other you will learn more about the Windows Internal Database (SSEE), how to dynamically generate T-SQL statements (i.e. how to massively detach and re-attach SQL Server databases) and how you can install 32-bit SQL Server Reporting Services on 64-bit machine.

Last but not least, in the last article of this chapter you will find a large set of different SQL-Server related tips focusing on the following areas:

- Solutions to common issues
- Basic string functions
- Performance-related tips
- Maintenance
- Miscellaneous

Windows Internal Database (SSEE)

(source: http://aartemiou.blogspot.com/2009/06/windows-internal-database-ssee.html)

The Windows Internal Database is a special variant of Microsoft SQL Server Express 2005-2012. It is included with Windows Server 2008/R2/2012, Windows Server Update Services (WSUS), Windows SharePoint Services (WSS) and other Windows Server tools. It is also referenced as SQL Server Embedded Edition (SSEE).

The Windows Internal Database is used by Active Directory, WSS, WSUS and other Windows Server services as their data storage. Some of its main characteristics are:

- It cannot be used as a regular SQL Server instance as it is intended to be only used by Windows Services/Programs.
- It cannot be removed by using the "Add or Remove Programs" tool because it does not appear in the list of currently installed programs (more info here). Note that it is not recommended to uninstall SSEE as it might affect the operation of Windows Services that use it.
- It only supports Windows Authentication.
- You can only connect to the instance using Named Pipes.

In some cases where you might need to access this special instance of SQL Server (i.e. for reducing the transaction log size of a database) you must use the Named Pipes protocol for doing so. Named Pipes can be enabled from Network Configuration in SQL Server Configuration Manager.

First of all you will need to install SQL Server 2005 Express Edition Management Studio or later (link). Then you will need to use the following connection properties:

- Server Type: **Database Engine**
- Server Name: **\\.\pipe\MSSQL$MICROSOFT##SSEE\sql\query**
- Authentication: **Windows Authentication**

***Note**: You have to be careful when accessing the Windows Internal Database as many Windows Services depend on it.

➔ ***Applies to: SQL Server 2005 or later.***

Dynamically Generating T-SQL Statements

(source: http://aartemiou.blogspot.com/2010/12/dynamically-generating-t-sql-statements.html)

In previous articles we talked about the undocumented stored procedures "sp_msforeachdb" and "sp_msforeachtable" which allow you to massively perform changes on all databases and tables within a SQL Server instance. However, massively performing changes on database

objects does not give you much control over the process and many times, even the experienced DBA or Developer might encounter problems if for example a database object is mistakenly included in the above process.

A method that allows you to still massively manage database objects but also gives you more control is to generate dynamic T-SQL. By using some of the catalog views in SQL Server 2005 or later you can easily do that.

The idea is to use a basic SELECT statement and then dynamically build the T-SQL expression that eventually will generate the desired T-SQL code.

Here are some examples (before running the code, right-click in the query window, then select "Results To" and finally "Results to Text"):

Example 1: See the physical files used for each database

```
SELECT
'USE '+ name +';'+' SELECT name,physical_name FROM SYS.DATABASE_FILES'
FROM SYS.databases;
GO
```

Listing 4.1: Generating a set of statements for retrieving physical files info for all DBs.

Example 2: Change the schema owner for all tables within a database

```
SELECT
'ALTER SCHEMA [NEW_SCHEMA_NAME] TRANSFER ' + s.Name + '.' + t.Name
FROM sys.tables t
INNER JOIN
sys.schemas s on t.schema_id = s.schema_id WHERE s.Name =
['OLD_SCHEMA_NAME'];
GO
```

Listing 4.2: Generating a set of statements for changing the schema owner for all DBs.

Example 3: Change the schema owner for all stored procedures within a database

```
SELECT
'ALTER SCHEMA [NEW_SCHEMA_NAME] TRANSFER ' + s.Name + '.' + sp.Name
FROM sys.Procedures sp
INNER JOIN
sys.Schemas s on sp.schema_id = s.schema_id WHERE s.Name =
['OLD_SCHEMA_NAME'];
GO
```

Listing 4.3: Generating a set of statements for changing the schema owner for all SPs.

Example 4: See all the views for each database

```
SELECT
'USE '+ s.name +';'+' SELECT * FROM sys.views' FROM SYS.databases s;
GO
```

Listing 4.4: Generating a set of statements for viewing all view for all DBs.

Example 5: Change the compatibility level for all the databases to 100 (SQL Server 2008)

```
SELECT 'ALTER DATABASE '+ name + ' SET COMPATIBILITY_LEVEL = 100'
FROM SYS.DATABASES;
GO
```

Listing 4.5: Generating a set of of statements for changing the compatibility level of all DBs.

Example 6: Set all the databases to READ ONLY mode

```
SELECT 'ALTER DATABASE '+ name + ' SET READ_ONLY WITH ROLLBACK IMMEDIATE'
FROM SYS.DATABASES;
GO
```

Listing 4.6: Generating a set of of statements for setting all DBs to Read-Only mode.

Example 7: Set all the databases to READ/WRITE mode

```
SELECT 'ALTER DATABASE '+ name + ' SET READ_WRITE WITH ROLLBACK IMMEDIATE'
FROM SYS.DATABASES;
GO
```

Listing 4.7: Generating a set of of statements for setting all DBs to Read/Write mode.

When using this technique along with the information retrieved from the catalog views, the possibilities are endless.

Yes, you still need to manually execute each generated T-SQL statement but the good thing is that the statements are being generated dynamically and by manually executing them, you have more control over the process.

* **Note:** The above sample code is intended only for demo purposes. Do not use it on Production systems. Whenever you massively modify database objects you need to be very careful with your data. Always backup your data.

➔ ***Applies to: SQL Server 2005 or later.***

▪ Massively Detaching and Re-attaching Databases in SQL Server

(source: http://aartemiou.blogspot.com/2013/08/massively-detaching-and-re-attaching.html)

There are cases where you might need to massively detach and re-attach all the databases in a SQL Server instance for any reason. In the case where you have 2-3 user databases on the instance it might not be an issue detaching and re-attaching the databases manually. However, imagine having over 100 databases. It would be not the easiest thing to go and detach/re-attach each database one-by-one (keep in mind that you have to specify the data/log files for each database when you are attaching it back to the instance).

In order to simplify things, I have developed a simple procedure that undertakes generating the necessary detach/attach scripts when you want to massively perform these actions on a SQL Server 2005 instance or later.

*** Note:** This procedure is presented in this article only for showing a different way of doing things. Always take backups of your data prior to any changes or actions.

The procedure for generating the scripts is the following:

1. Run the "**Attach DDL Statements Generation Scrip**"
2. Run the "**Detach DDL Statements Generation Scrip**t"

If you want to proceed and detach all user databases you can then execute the set of DDL statements generated by the "Detach DDL Statements Generation Script".

If you want to attach the databases on the same instance or any other SQL Server instance on the same server you can then execute the set of DDL statements generated by the "Attach DDL Statements Generation Scrip".

You can view the scripts below:

Attach DDL Statements Generation Scrip (on next page):

```
--
--Attach DDL Statements Generation Scrip
--Author: Artemakis Artemiou, SQL Server MVP
--
print '--'
print '--Script for Attaching all DBs in a SQL Server Instance'
print '--'
print ''
SET NOCOUNT ON

DECLARE @dbname nvarchar(128)

DECLARE DBList_cursor CURSOR FOR
select [name] from master.sys.databases where database_id > 4;

OPEN DBList_cursor

 FETCH NEXT FROM DBList_cursor
 INTO @dbname;

 WHILE @@FETCH_STATUS = 0
 BEGIN

  declare @attach_TSQL_script varchar(max);
  set @attach_TSQL_script='';
  set @attach_TSQL_script=@attach_TSQL_script+'CREATE DATABASE ' + @dbname +' ON ' ;

  declare @tsql varchar(max),@filename varchar(max);
  set @tsql='DECLARE DBFiles_cursor CURSOR FOR select [filename] from '+ @dbname +
'.sys.sysfiles';
  execute (@tsql);

  PRINT '--'+@dbname;

  OPEN DBFiles_cursor
  FETCH NEXT FROM DBFiles_cursor INTO @filename;

  WHILE @@FETCH_STATUS = 0
  BEGIN
   set @attach_TSQL_script=@attach_TSQL_script+' (FILENAME = '''+ @filename +'''),';
   FETCH NEXT FROM DBFiles_cursor INTO @filename
  END

 set @attach_TSQL_script=SUBSTRING(@attach_TSQL_script,0,len(@attach_TSQL_script));
 set @attach_TSQL_script=@attach_TSQL_script+' FOR ATTACH;';

 PRINT @attach_TSQL_script;
 PRINT '';

CLOSE DBFiles_cursor;
DEALLOCATE DBFiles_cursor;

FETCH NEXT FROM DBList_cursor
INTO @dbname;

END

CLOSE DBList_cursor;
DEALLOCATE DBList_cursor;
```

Listing 4.8: Generating attach DDL scripts for all DBs in a SQL Server Instance.

Detach DDL Statements Generation Scrip

```
--
--Detach DDL Statements Generation Scrip
--Author: Artemakis Artemiou, SQL Server MVP
--

PRINT '--';
PRINT '--Script for Detaching all DBs in a SQL Server Instance';
PRINT '--';
PRINT '';

SET nocount ON;

DECLARE @dbname nvarchar(128);
DECLARE dblist_cursor CURSOR FOR
  SELECT [name]
  FROM   master.sys.databases
  WHERE  database_id > 4;

OPEN dblist_cursor
FETCH next FROM dblist_cursor INTO @dbname;

WHILE @@FETCH_STATUS = 0
  BEGIN
    print 'EXEC sp_detach_db ''' + @dbname + ''', ''true'';';
    FETCH next FROM dblist_cursor INTO @dbname;
  END

CLOSE dblist_cursor;
DEALLOCATE dblist_cursor;
```

Listing 4.9: Generating detach DDL scripts for all DBs in a SQL Server Instance.

Download the scripts

You can download the scripts from the links below:

- Attach DDL Statements Generation Scrip
- Detach DDL Statements Generation Scrip

➔ ***Applies to: SQL Server 2005 or later.***

▪ Installing 32-bit SQL Server 2005 Reporting Services on a 64-bit machine/Windows OS

(source: http://aartemiou.blogspot.com/2009/05/installing-32-bit-sql-server-2005.html)

Many fellow community members ask whether it is possible to install 32-bit SQL Server 2005 Reporting Services on a 64-bit Computer/Windows OS while keeping the 64-bit version of the

rest of SQL Server features. The answer is yes.

Consider the following real-life scenario that happened to me a few years ago.

I had installed SQL Server 2005 on a 64-bit Windows Server 2003 OS. This was done within the context of setting up a test environment for an ASP .NET application. As the OS was a 64-bit architecture I had installed the 64-bit version of SQL Server as well.

Though, the ASP .NET application was compiled on .NET Framework 1.1 and IIS 6.0 was set up to use this version of the .NET Framework. To this end, when the installation of the reporting services started, I got the message that it was not possible to install the 64-bit version of the SQL Server 2005 Reporting Services because of using .NET Framework 1.1 for IIS 6.0.

Though, SQL Server did not disappoint me. With some simple steps, it was very easy to install the 32-bit version of the Reporting Services while keeping installed the 64-bit version of the rest of SQL Server features (i.e. Database Engine, Analysis Services, etc.).

The following MSDN Link provides excellent instructions on how to perform this:
http://msdn.microsoft.com/en-us/library/ms143293(SQL.90).aspx

*** Note:** The procedure is almost identical for **SQL Server 2008** with the difference that when you are going to install Reporting Services, before starting the installation process, from within SQL Server 2008 Installation Center, click on "**Options**" and select the "**x86" Processor Type**", then click on "**Installation**" and after selecting "**New SQL Server stand-alone installation or add features to an existing installation**" install only Reporting Services.

➔ ***Applies to: SQL Server 2005 or later.***

▪ Useful SQL Server Knowledge

(source: http://aartemiou.blogspot.com/2011/12/useful-sql-server-knowledge-set-1.html)

In this article you can find different tips and T-SQL scripts organized in the following categories:

A. Solutions to common issues
B. Basic string functions
C. Performance-related tips
D. Maintenance
E. Miscellaneous

A: Solutions to common issues

1. Resolving the "Divide by zero" error (by example)

```
DECLARE @denominator INT
SET @denominator = 0
SELECT 1 / ISNULL(NULLIF(@denominator, 0), 1);
GO
```

Listing 4.10: Resolving the "Divide by zero" error.

2. Handling NULL and empty values

```
----Step 1: Create the IsEmpty user-defined function
CREATE FUNCTION IsEmpty
(
@input AS VARCHAR(250),
@newValue VARCHAR(250)
)
RETURNS VARCHAR(250)
AS
BEGIN
-- First handle the case where the input value is a NULL
DECLARE @inputFiltered AS VARCHAR(250);
SET @inputFiltered = ISNULL(@input, '');


-- The main logic goes here
RETURN (CASE RTRIM(LTRIM(@inputFiltered)) WHEN '' THEN RTRIM(LTRIM(@newValue))
ELSE RTRIM(LTRIM(@inputFiltered)) END);


END
GO

----Step 2: Usage
SELECT dbo.IsEmpty(@column_to_check, @new_value);
GO
```

Listing 4.11: Handling NULL and empty values.

3. Handling the error "A transport-level error has occurred when sending the request to the server"

If the problem occurs in a SSMS Query Window, just try again to run your set of statements and a new connection will be re-initiated.

4. Handling the error "The conversion of a char data type to a datetime data type resulted in an out-of-range datetime value"

Change the default language to "us_english" for the given SQL Server login:

```
USE [master];
GO
ALTER LOGIN "LOGIN_NAME" WITH DEFAULT_LANGUAGE = us_english;
GO
```

Listing 4.12: Handling the out-of-range datetime value error.

Best practice: Always use the ISO date format in your data applications/T-SQL scripts: **YYYY-MM-DD**

5. Handling the error "The multi-part identifier ... could not be bound"
Be careful with the use of subqueries and table aliases. Don't forget to reference the correct table aliases in your T-SQL code.

Also, keep in mind that subqueries can only provide their results to their outer queries and not references to the subqueries' tables.

6. Handling the error "String or binary data would be truncated"
Either use an adequate size for the table columns in which the data is inserted or cast the data by removing redundant characters. I suggest the first approach.

7. Handling the error "Error converting data type varchar to float"

```
----Step 1: Create the Varchar2Float user-defined function
CREATE FUNCTION [dbo].[Varchar2Float]( @inputString VARCHAR(50) )
RETURNS FLOAT
AS
BEGIN
--Prepare the string for casting/conversion
SET @inputString = REPLACE(@inputString, '.', '');
SET @inputString = REPLACE(@inputString, ',', '.');

--Perform the conversion and return the result
RETURN CAST(@inputString AS FLOAT);
END
GO

----Step 2: Usage
SELECT dbo.Varchar2Float(@value);
GO
```

Listing 4.13: Handling the error "Error converting data type varchar to float".

8. Handling the error "Database [Database_Name] cannot be upgraded because it is read-only or has read-only files"
Make sure that the user account on which the SQL Server instance database engine is running has full access to the database files. Also, if you are working in Windows Vista or later, try running SSMS as Administrator.

B. Basic string functions

1. Returns @length characters from @expression starting from @start_index

```
SELECT SUBSTRING(@expression, @start_index, @length);
GO
```

Listing 4.14: Using the SUBSTRING string function.

2. Finds the given @pattern in the @string and replaces it with the @replacement_string

```
SELECT REPLACE(@string, @pattern, @replacement_string);
GO
```

Listing 4.15: Using the REPLACE string function.

3. Returns the size of @string in terms of number of characters

```
SELECT LEN(@string);
GO
```

Listing 4.16: Getting the length of a string.

4. Returns the first @num_chars characters of the @string counting from the left

```
SELECT LEFT(@string, @num_chars);
GO
```

Listing 4.17: Using the LEFT string function.

5: Returns the first @num_chars characters of the @string counting from the right

```
SELECT RIGHT(@string, @num_chars);
GO
```

Listing 4.18: Using the RIGHT string function.

6: Removes the leading blank spaces

```
SELECT LTRIM(@expression);
GO
```

Listing 4.19: Removing leading blank spaces in a string.

7: Removes the trailing blank spaces

```
SELECT RTRIM(@expression);
GO
```

Listing 4.20: Removing trailing blank spaces in a string.

C. Performance-related tips

1. Avoiding locking when reading data (however, dirty reads are allowed)

```
SELECT [columnName]
FROM [tableName] WITH (NOLOCK);
GO
```

Listing 4.21: Using the NOLOCK table hint.

2. Rebuilding indexes in SQL Server 2005 or later

```
USE [DATABASE_NAME];
ALTER INDEX [INDEX_NAME] ON [SCHEMA.TABLE]
REBUILD WITH (FILLFACTOR=[FILL_FACTOR_VALUE_BETWEEN_0_100], ONLINE=[ON|OFF]);
GO
```

Listing 4.22: Rebuilding a specific index with using parameters.

```
USE [DATABASE_NAME];
ALTER INDEX ALL ON [SCHEMA.TABLE]
REBUILD WITH (FILLFACTOR=[FILL_FACTOR_VALUE_BETWEEN_0_100], ONLINE=[ON|OFF]);
GO
```

Listing 4.23: Rebuilding all indexes in a table with using parameters.

3. Rebuilding all the indexes in a database

```
USE [DATABASE_NAME];
GO
EXEC sp_MSforeachtable @command1="print '?'", @command2="ALTER INDEX ALL ON ?
REBUILD WITH (ONLINE=ON)";
GO
```

Listing 4.24: Rebuilding all indexes online with keeping the default fill factor for each index.

```
USE [DATABASE_NAME];
GO
EXEC sp_MSforeachtable @command1="print '?'", @command2="ALTER INDEX ALL ON ?
REBUILD WITH (ONLINE=OFF)";
GO
```

Listing 4.25: Rebuilding all indexes offline with keeping the default fill factor for each index.

```
USE [DATABASE_NAME];
GO
EXEC sp_MSforeachtable @command1="print '?'", @command2="ALTER INDEX ALL ON ?
REBUILD WITH (FILLFACTOR=[FILL_FACTOR_PERC],ONLINE=ON)";
GO
```

Listing 4.26: Rebuilding all indexes online with specifying the fill factor.

```
USE [DATABASE_NAME];
GO
EXEC sp_MSforeachtable @command1="print '?'", @command2="ALTER INDEX ALL ON ?
REBUILD WITH (FILLFACTOR=[FILL_FACTOR_PERC],ONLINE=OFF)";
GO
```

Listing 4.27: Rebuilding all indexes offline with specifying the fill factor.

4. Updating database tables without causing blocking

```
UPDATE [TABLE_NAME] WITH (READPAST)
SET ...
WHERE ...
GO
```

Listing 4.28: Updating tables with using table hints.

D. Maintenance

1. Shrinking an entire database

```
DBCC SHRINKDATABASE([DBName],[PercentageOfFreeSpace]);
GO
```

Listing 4.29: Shrinking a database.

2. Truncating an entire database

```
DBCC SHRINKDATABASE([DBName],TRUNCATEONLY);
GO
```

Listing 4.30: Releasing back to the OS the occupied space at the end of the DB files.

3. Shrinking a data/log file

```
USE [DBName];
GO
DBCC SHRINKFILE ([Data_Log_LogicalName],[TargetMBSize]);
GO
```

Listing 4.31: Shrinking a data or log file with specific target in MB.

4. Truncating a data/log file

```
USE [DBName];
GO
DBCC SHRINKFILE ([Data_Log_LogicalName],TRUNCATEONLY);
GO
```

Listing 4.32: Releasing back to the OS the occupied space at the end of the data or log file.

5. Renaming a Windows login

```
ALTER LOGIN "[Domain or Server Name]\[Windows Username]"
WITH NAME="[New Domain or New Server Name]\[Windows Username]";
GO
```

Listing 4.33: Renaming a Windows login.

6. Renaming a SQL Server login

```
ALTER LOGIN "[SQL Server Login Name]"
WITH NAME="[New SQL Server Login Name]";
GO
```

Listing 4.34: Renaming a SQL Server login.

7. Creating Logins for orphaned SQL Server users

```
USE [DBName];
GO
EXEC sp_change_users_login 'Auto_Fix', '[UserName]', NULL, '[Password]';
GO
```

Listing 4.35: Creating Logins for orphaned SQL Server users

8. Changing the Database Owner in a SQL Server Database (SQL Login)

```
USE [DBName];
GO
EXEC sp_changedbowner '[SQL_Login_Name]';
GO
```

Listing 4.36: Changing the Database Owner in a SQL Server Database (SQL Login)

9. Changing the Database Owner in a SQL Server Database (Windows Login)

```
USE [DBName];
GO
EXEC sp_changedbowner '[DomainName\UserName]';
GO
```

Listing 4.37: Changing the Database Owner in a SQL Server Database (Windows Login)

10. Backing up a Database in a Network Folder (creating the destination media)

```
USE [master];
GO
EXEC sp_addumpdevice 'disk',
'NetworkDeviceName','\\serverName\backupFolder\BackupFileName.bak';
GO
```

Listing 4.38: Creating the destination media for backing up a database in a network folder.

Now you can use the above "disk" device to backup databases onto the network location you specified.

E. Miscellaneous

1. Gets basic information on the current SQL Server instance

```
SELECT
SERVERPROPERTY('ProductVersion') AS ProductVersion,
SERVERPROPERTY ('ProductLevel') AS ProductLevel,
SERVERPROPERTY ('Edition') AS Edition,
SERVERPROPERTY('MachineName') AS ServerName,
SERVERPROPERTY('ServerName') AS Server_and_Instance_Names;
GO
```

Listing 4.39: Retrieving SQL Server instance-related information.

2. Gets basic table index information

```
EXEC sp_helpindex 'schema.table_name';
GO
```

Listing 4.40: Retrieving table index information.

3. Connecting to Windows Internal Database (SSEE)
From within SSMS (Express Edition works as well):

- Server Type: Database Engine
- Server Name: \\.\pipe\MSSQL$MICROSOFT##SSEE\sql\query
- Authentication: Windows Authentication

***Note:** Make sure that the Named Pipes protocol is enabled.

➔ ***Applies to: SQL Server 2000 (partially), 2005 or later (fully).***

▪ Summary

In this chapter you learned what the Windows Internal Database (SSEE) is and how you can access it. Additionally you learned how to dynamically generate T-SQL statements as well as how you can install 32-bit SQL Server Reporting Services on 64-bit machine. Furthermore, you learned various T-SQL tips having to do with different SQL Server topics such as: solutions to common issues, basic string functions, performance tuning and maintenance. The next chapter discusses how you can overcome specific errors you might get due to network problems, permissions, missing service packs/patches etc.

CHAPTER 5

Error Handling

IN THIS CHAPTER:

- Could not load file or assembly 'Microsoft.SqlServer.Smo, Version=10.0.0.0, ...
- Creating an instance of the COM component with CLSID…
- There was an unexpected failure during the setup wizard
- No global profile is configured. Specify a profile name in the @profile_name parameter
- Database [Database_Name] cannot be upgraded because it is read-only or has read-only files
- Saving maintenance plan failed
- Exclusive access could not be obtained because the database is in use
- A transport-level error has occurred when sending the request to the server

When working with an enterprise system, in most of the times, the possibilities are endless. SQL Server allows its users to do almost anything that has to do with data storage, organization and processing. SQL Server users are presented with a plethora of features, tools and technologies shipped with SQL Server that they can be used to transform raw data into knowledge.

However, even though all these technologies and tools are available, their proper operation depends on many external factors too such as:

- A stable infrastructure (server, storage, network).
- Proper maintenance in terms of installing the latest patches/service packs.
- Valid user input.
- Valid permissions.

In cases where the above factors are not satisfied, there is the possibility of receiving an error message when trying to perform an action that is directly affected by those factors (i.e. trying to insert records into a database/table where your login does not have write access, or trying to access a database and the network connection is abnormally terminated, etc.). Usually these error messages are self-explanatory thus helping the user to understand what needs to be done in order to resolve the issue.

In some other cases, the error messages can be more generic thus requiring further research on behalf of the user on what the exact problem is and how can it be resolved. In these cases you can find online material that can help you resolving the problem or seek professional services.

This chapter presents different such cases and proposes ways of overcoming such issues.

Could not load file or assembly 'Microsoft.SqlServer.Smo, Version=10.0.0.0, ...

(source: http://aartemiou.blogspot.com/2012/05/could-not-load-file-or-assembly.html)

At some point in the past I have experienced an issue which had to do with a third-party application that was trying to retrieve some meta information from a specific SQL Server Instance. It was actually trying to retrieve the names of all the databases contained within that instance.

However it was returning the following error message (app code excluded):
[.....] Received the following from the MS SQL server: Could not load file or assembly 'Microsoft.SqlServer.Smo, Version=10.0.0.0, Culture=neutral, PublicKeyToken=89845dcd8080cc91' or one of its dependencies. The system cannot find the file specified.

SMO stands for Shared Management Objects and it is basically an API that can be used by third-party applications for programmatically managing all aspects of SQL Server.

As the above message was quite explanatory, I directly checked the Global Assembly Cache (GAC) under "**c:\windows\assembly**" to see if the assembly (DLL) was listed there. I immediately noticed that it was not.

In such cases you have two options:

1. Run the installer that registers the missing assemblies to the GAC.
2. Use Gacutil.exe to manually register the assembly after you get the relevant dll.

The simplest thing to do is to run the installer. From the error description, version 10.0.0.0 of SMO is the SQL Server 2008 version so I just had to download the proper package and install it.

The package can be found in Microsoft SQL Server 2008 Feature Pack, August 2008. You can download it here.

After visiting the above link, look for "**Microsoft SQL Server 2008 Management Objects**". The available packages are the following:

- X86 Package (SharedManagementObjects.msi)
- X64 Package (SharedManagementObjects.msi)
- IA64 Package (SharedManagementObjects.msi)

I downloaded and installed the proper package, verified that the assembly was registered in GAC, and tried again. The application worked, successfully listing the names of the databases in the instance.

▪ Creating an instance of the COM component with CLSID…

(source: http://aartemiou.blogspot.com/2011/11/creating-instance-of-com-component-with.html)

If you ever encounter issues when trying to create, add, or edit steps for a SQL Server Agent job in SQL Server Management Studio 2008 R2 don't worry, there is a solution.

The error popup you might get is the following:

Figure 5.1: Error message when trying to view/add/edit a SQL Server Agent job step.

The fastest thing you can do, is restarting SSMS and try again.

Additionally, there is a fix for this issue. You can find it here.

The fix applies to the following products:

- Microsoft SQL Server 2008 R2 Developer
- Microsoft SQL Server 2008 R2 Enterprise
- Microsoft SQL Server 2008 R2 Workgroup
- Microsoft SQL Server 2008 R2 Datacenter
- Microsoft SQL Server 2008 R2 Standard

▪ There was an unexpected failure during the setup wizard

(source: http://aartemiou.blogspot.com/2012/11/there-was-unexpected-failure-during.html)

Consider the following scenario: You are trying to install SQL Server 2005, and then you stop the installation, let's say because you got a warning on a prerequisite that you would like to fix before proceeding. However, in the case you abort the installation, fixed the problem and try to

install SQL Server again you now get the following error message:

There was an unexpected failure during the setup wizard. You may review the setup logs and/or click the help button for more information.

As I have faced this issue some time ago, this is the solution I have concluded to:

1. Delete Everything in the folder "C:\Program Files\Microsoft SQL Server\90\Setup Bootstrap"
2. Enter the registry (Start→Run→Regedit)
3. Delete all the SQL Server 2005 "Setup Bootstrap" keys in "HKEY_CLASSES_ROOT\Installer\Assemblies". **Example: "HKEY_CLASSES_ROOT\Installer\Assemblies\C:|Program Files (x86)|Microsoft SQL Server|90|Setup Bootstrap|BPA|bin|BPAClient.dll"**
4. Restart the machine
5. Start again the installation of SQL Server 2005

The philosophy behind this solution is that you actually need to "clean" the machine from any "pending" installation setup settings/files/registry keys/etc. of SQL Server 2005 before you attempt to start a new installation.

The reason you need to do this is because when you start a new installation, if it finds remains of a previous installation it "reads" it and proceeds based on those.

***Important Note:** Be extremely careful when you are modifying the Windows Registry. Always take a backup before you do anything and of course always remember what's written on Registry's door: "**Enter at your own risk!"**

▪ No global profile is configured. Specify a profile name in the @profile_name parameter

(source: http://aartemiou.blogspot.com/2011/02/no-global-profile-is-configured-specify.html)

Whenever you set up a maintenance plan, it is always a good idea to add in the end the "Notify Operator" task in order to get notified on the outcome of the maintenance plan's execution.

Here's a screenshot of a simple maintenance plan I created in the past having the Notify Operator" task as well:

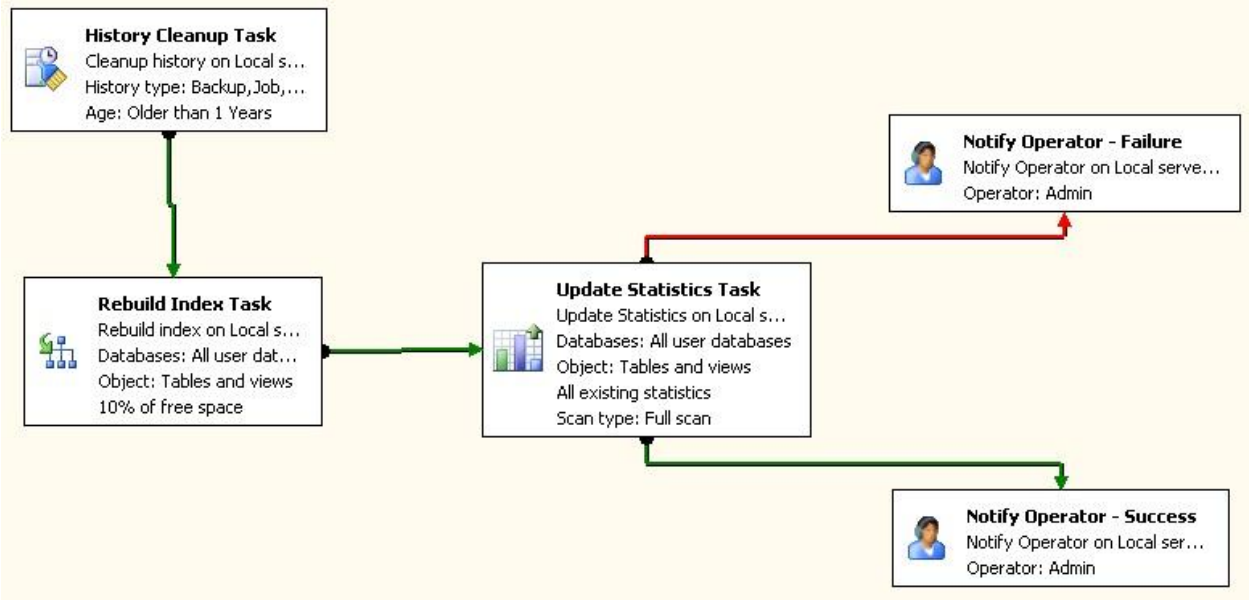


Figure 5.2: Simple maintenance plan with the "Notify Operator" option included.

Even though the first three tasks were executed successfully when I ran the plan, when I checked the history of the job that undertakes the execution of the maintenance plan, there was an error message on the last task that is the "Notify Operator" task.

A part of the error message is the following:

"No global profile is configured. Specify a profile name in the @profile_name parameter."

So, what does the above error message tell us? It actually says that upon the task's execution, it cannot find a Database Mail profile in order to use it for sending the email notification.

To overcome this problem, you can set one of the available Database Mail profiles as the **default** one.

In SQL Server 2005 you can do this in the following way through SSMS:

1. Management → Database Mail (double-click or right-click - Configure Database Mail).
2. Click on Next.
3. Manage profile security.
4. Click on Next.
5. Then you will see a list with all the available database mail profiles. You have the option to select one of these profiles to be the Default Profile (be selecting 'Yes' in the combo box).
6. Click on Next.
7. Click on Finish.

***Note:** The above assumes that you have at least one Database Mail profile up and working.

Database [Database_Name] cannot be upgraded because it is read-only or has read-only files

(source: http://aartemiou.blogspot.com/2011/03/database-databasename-cannot-be.html)

At some time in the past, I was trying to attach some databases on a SQL Server 2005 instance. The database files were copied over the network and located on a drive on the DBMS server.

Though, while I was trying to attach the databases I was getting an error message of the following type:

Msg 3415, Level 16, State 3, Line 1

Database **[database_name]** cannot be upgraded because it is read-only or has read-only files. Make the database or files writeable, and rerun recovery.

Msg 1813, Level 16, State 2, Line 1

Could not open new database **[database_name]**. CREATE DATABASE is aborted.

There are two ways for resolving this.

The first and simplest way is to exit SSMS and run it again as Administrator. This will allow you to use the data/log files with full permissions via SSMS.

A second way, when you cannot run SSMS as Administrator, would be the following:

1. Check the permissions of the database files and ensure that they are **not** read-only.
2. Also, ensure that the service user account running the SQL Server instance has full access on the files.
3. Provide full access to "**Everyone**" on all the database files, attach them, and run the original operation you were trying earlier to run.
4. Restore the original access rights on the database files earlier modified.

Saving maintenance plan failed

(source: http://aartemiou.blogspot.com/2009/10/saving-maintenance-plan-failed.html)

If you encounter the above error message when trying to save a maintenance plan you created using the wizard in SQL Server 2005 Management Studio (SSMS), the first thing you should try is to check if the latest service pack is installed and if it is not then install it.

You can run the following T-SQL script for finding out which service pack is currently installed on your SQL Server 2005 instance:

```
SELECT
SERVERPROPERTY('productversion'),
SERVERPROPERTY ('productlevel'),
SERVERPROPERTY ('edition');
GO
```

Listing 5.1: Retrieving SQL Server service pack information.

However, if you are still experiencing the same problem, even in the case you have installed the latest service pack, then you can try running the T-SQL script called "**sysdbupg.sql**" which can be found in the SQL Server installation directory under the subfolder "**install**".

This script upgrades the system database stored procedures from the RTM level. Databases which may be upgraded are:

- master
- model
- msdb

*** Note**: Make sure before trying to execute the above script, to backup the abovementioned system databases!

Applies to: SQL Server 2005.

▪ Exclusive access could not be obtained because the database is in use

(source: http://aartemiou.blogspot.com/2009/03/exclusive-access-could-not-be-obtained.html)

A very useful feature of SQL Server Database Engine is the way it lets the DBA to control connections. To this end, you can explicitly control connections with specific commands and methods. Even though this is not usually needed, there are some cases where you might need to make use of this feature. One such a case is when you are presented with the error:

"Exclusive access could not be obtained because the database is in use"

In SQL Server you might be presented with the above error message in the case you are trying to perform an operation on a database that requires exclusive access. One such operation is when you want to urgently restore a database.

Exclusive access is when an operation needs to run on a database (or another SQL Server object) without any other operations/sessions accessing the same database/object at the same time.

So, what can you do when you get this error message? You still need to perform the RESTORE

right?

Here's a common way of doing that:

1. Set the database in SINGLE_USER mode and forcibly terminate other connections:

```
USE [master];
GO
ALTER DATABASE [DATABASE_NAME]
SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
GO
```

Listing 5.2: Setting the database to SINGLE_USER mode.

The above command will set the database in **SINGLE_USER MODE** and additionally the **ROLLBABK IMMEDIATE** termination option will roll back all the incomplete transactions. It will also disconnect any other connections to the database.

***Note:** After issuing the above command, you will only be able to use a single connection to the database. So, if you are already using a query window and thus you are connected to the target database, either restore the database by using the RESTORE T-SQL command or close the query window and restore the target database by using the Restore Database Wizard from SQL Server Management Studio.

2. Restore the database.

3. Set the database back to **MULTI_USER** mode:

```
USE [master];
GO
ALTER DATABASE [DATABASE_NAME]
SET MULTI_USER;
GO
```

Listing 5.3: Setting the database to MULTI_USER mode.

Here's an alternative way of doing the above:

1. If there are any SQL Server logins granted any access on the target database then disable them:

```
ALTER LOGIN [LOGIN_NAME] DISABLE;
GO
```

Listing 5.4: Disabling a login.

***Note:** Even though now you have disabled the login(s) there still might exist active connections to the database.

2. To find and terminate existing connections on the target database perform the following:

```
-- Query that returns all the SPIDs (process IDs)/sessions
-- established to a given database:
SELECT spid,loginame,login_time,program_name
FROM [master]..sysprocesses
WHERE DBID=DB_ID('DATABASE_NAME');
GO
```

Listing 5.5: Retrieving the active sessions/connections to a database.

By collecting the above SPIDs you get the sessions that must be forcibly terminated (be careful not to forcibly terminate your own session! :) By including the columns **loginame**, **login_time** and **program_name** in the above query, you are able to identify and exclude the spid that belongs to your session (that is, the DBA session).

You can terminate a session by executing the T-SQL Statement:

```
KILL @SPID;
GO
```

Listing 5.6: Terminating a SQL Server session.

***Note 1:** Replace @SPID with the appropriate SPID number.
***Note 2:** You can even create a user-defined function undertaking this task.

3. Set the database in **SINGLE_USER** mode:

```
USE [master];
GO
ALTER DATABASE [DATABASE_NAME]
SET SINGLE_USER;
GO
```

Listing 5.7: Setting the database to SINGLE_USER mode.

***Note:** After issuing the above command, you will only be able to use a single connection to the database. So, if you are already using a query window connected to the target database, either restore the database by using the RESTORE T-SQL command or close the query window and restore the target database by using the Restore Database Wizard from SQL Server Management Studio.

4. Restore the database.

5. Set the database back to **MULTI_USER** mode:

```
USE [master];
GO
ALTER DATABASE [DATABASE_NAME]
SET MULTI_USER;
GO
```

Listing 5.8: Setting the database to MULTI_USER mode.

Remarks and Considerations

In the case of a RESTORE operation I would personally prefer the first method as it is simpler. With a single T-SQL Statement you set the database in SINGLE_USER MODE and also terminate all the active connections immediately and roll back all the incomplete transactions.

The second method is more preferable in cases where more "elegant" session control is required. This method allows the DBA to terminate sessions one-by-one explicitly instead of the first method which massively terminates all connections.

Choosing one of the two ways I guess that relies to the urgency of the RESTORE operation, the judgment of the DBA and also any Business Policies that might stand for such cases.

***Important Note:** Because both the abovementioned methodologies have to do with forcibly terminating connections to a database in SQL Server, you must be extremely careful when using them as you might accidentally cause severe data loss when applying them inappropriately and without the necessary authorization from the Management.

➔ ***Applies to: SQL Server 2000 (partially), 2005 or later (fully).***

▪ A transport-level error has occurred when sending the request to the server

(source: http://aartemiou.blogspot.com/2009/02/transport-level-error-has-occurred-when.html)

This post's purpose is to explain the above connectivity error and ways of resolving it.

In some cases, typically when network problems occur, you might get the following error in SQL Server:

A transport-level error has occurred when sending the request to the server. (provider: Shared Memory Provider, error: 0 - No process is on the other end of the pipe.)

As the error message indicates, there is a connectivity problem with a previously opened session in SQL Server.

For example, consider that you have a query window opened. This query window is connected to a database (the default database is "master"). To this end, there is an established connection to that database that is a **session**. At a lower level in SQL Server, there is a **spid** representing this session.

If this spid is forcibly terminated (i.e. by using the SQL Server KILL function, or after an unexpected network problem) and though you are still trying to use this session (i.e. execute a query) you will most probably get the above connectivity error.

A solution in this case, is to close the query window and open a new one or re-run the operation (i.e. T-SQL script). To this end a new session will be opened and will be assigned a new spid. After that, you will be able to use the database properly.

The above connectivity error might arise not only when using the query window, but other SQL Server services that might access SQL Server objects as well. It is a general connectivity error and the best solution for resolving it is to start a new session in order for the connectivity to the specific SQL Server object(s) to be restored/re-initiated.

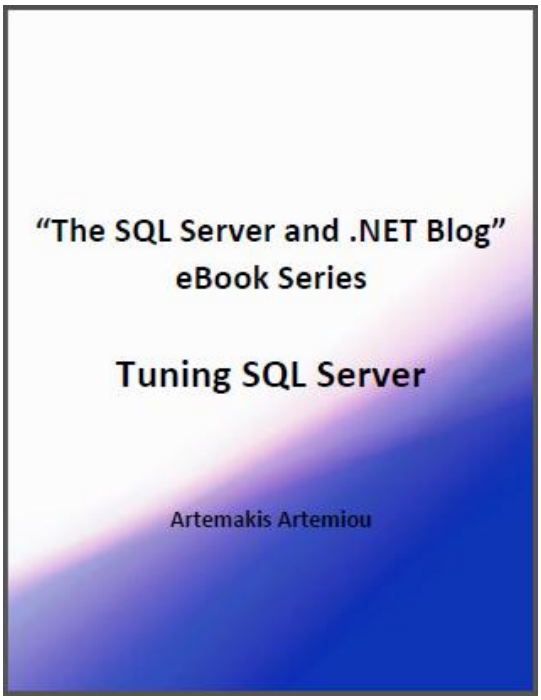
Summary

In this chapter you learned how to overcome specific errors you may encounter when working with SQL Server that might be caused due to different external factors such as: network problems, missing permissions, missing service packs/patches etc.

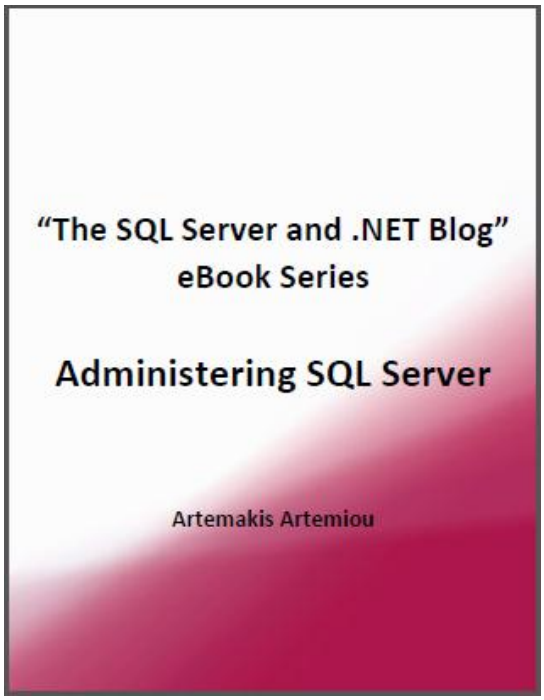
List of Listings

List of Figures

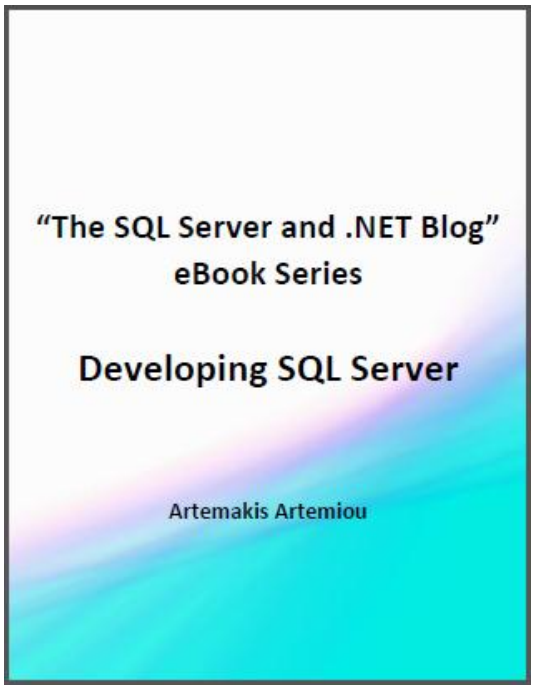
- The SQL Server and .NET eBook Series -



Release Date: October 1, 2013
http://sqlbooks.aartemiou.com/books/tuning/

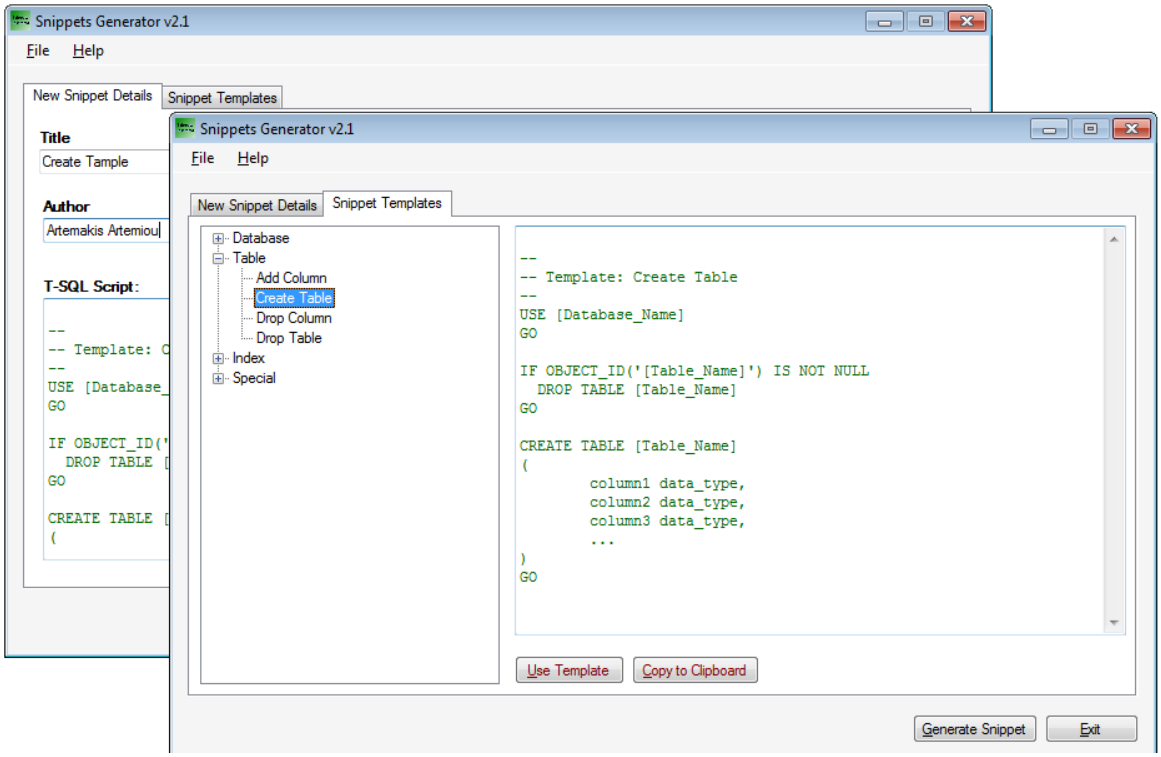


Release Date: December 2, 2013
http://sqlbooks.aartemiou.com/books/administration/

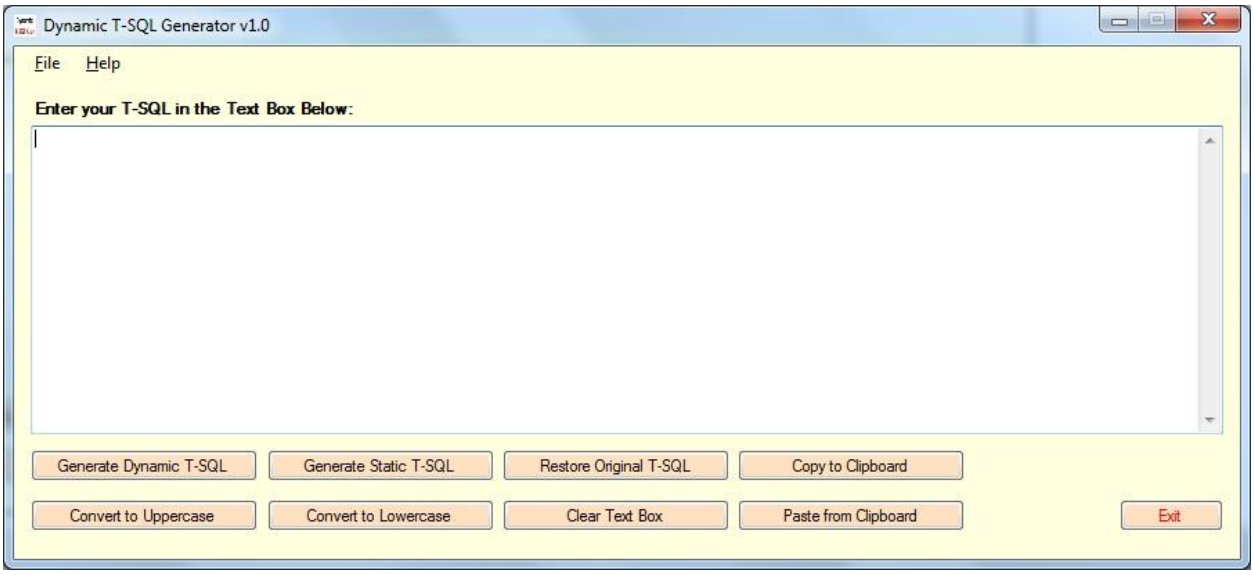


To be released in 2014
http://sqlbooks.aartemiou.com/books/development/

- CodePlex Projects -

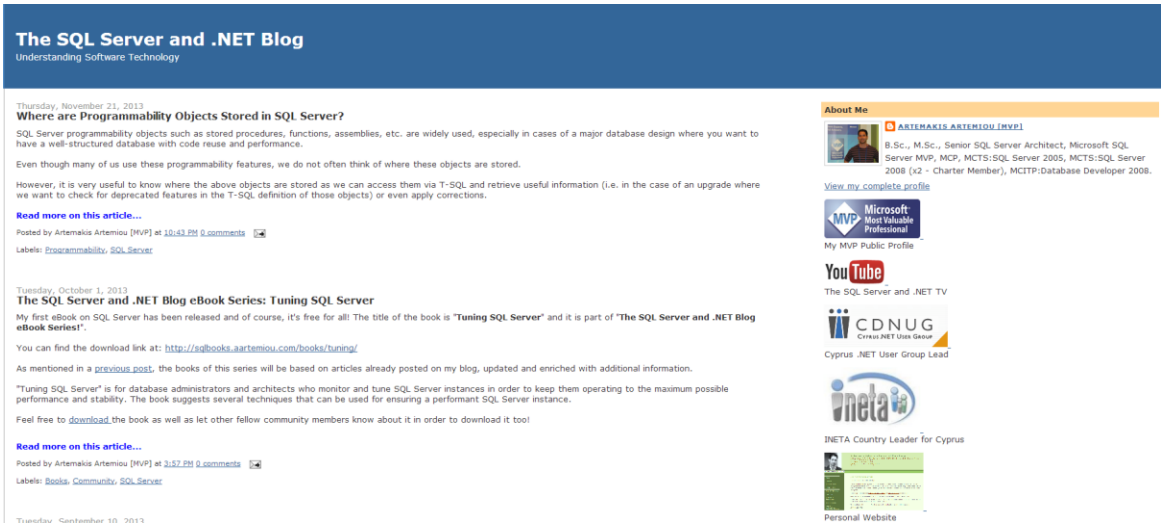


Snippets Generator: http://snippetsgen.codeplex.com/

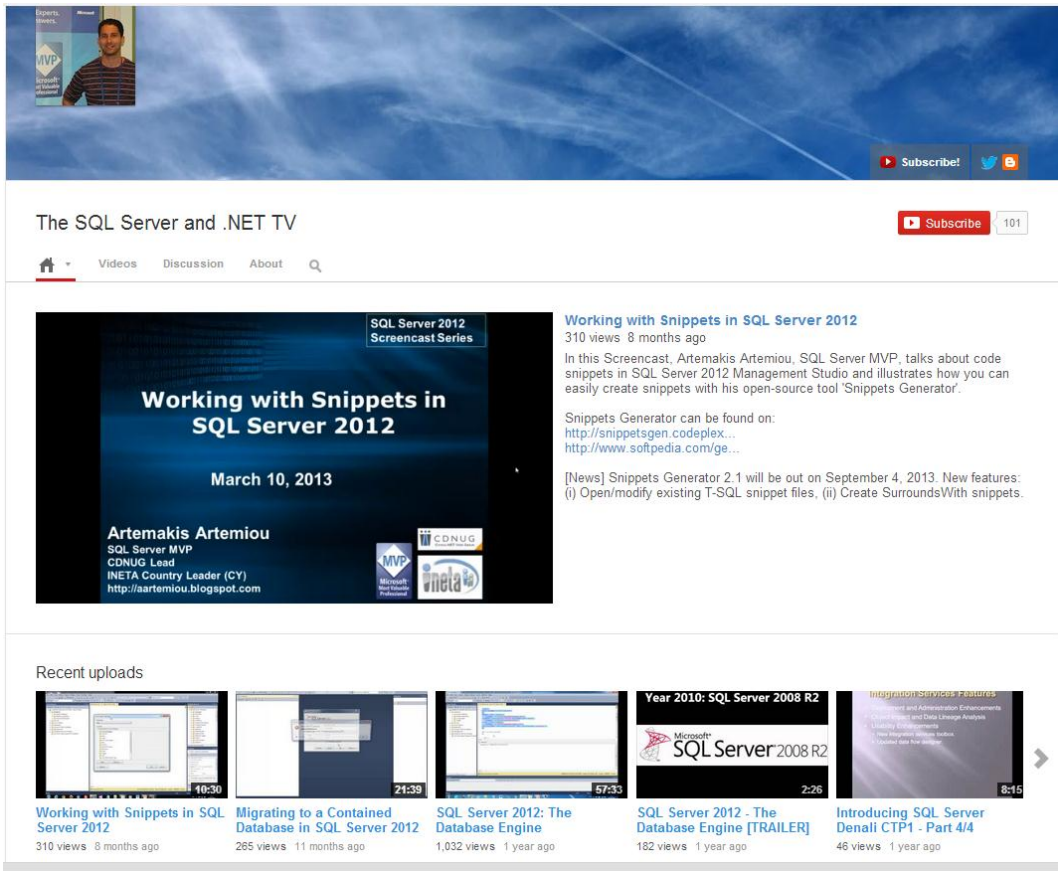


Dynamic T-SQL Generator: http://dynamicsqlgen.codeplex.com/

- Blog/TV -



The SQL Server and .NET Blog: http://aartemiou.blogspot.com/



The SQL Server and .NET TV: http://www.youtube.com/sqlserverdotnetblog/