Sitecore CMS 6.5 - 7.2

DMS Performance Tuning Guide for SQL Server

A system administrator's guide to optimizing the performance of Sitecore DMS

Table of Contents

Chapter 1

How To Use This Guide

The DMS Tuning Guide is designed to help you improve the performance of your DMS 2.x implementation. The guide is designed as a set of tasks, that are listed in order of importance.

Each task contains a description of symptoms, tuning recommendations and additional analysis.

1.1 Using the Analytics Database Manager

You can use the Analytics Database Manager (ADM) as an alternative means of diagnosing and performing many of the the tuning task presented in this guide.

If a task can be performed using the ADM, the task's title is marked with (ADM). For example - SQL Server Index Fragmentation Level (ADM).

You can download the ADM from the Sitecore Marketplace, where you can find the correct version to use for your implementation, along with a full set of documentation on how to use the tool — Analytics Database Manager.

Chapter 2

The Basics

The tasks presented in this chapter are the basic tuning procedures that provide the greatest impact on performance.

This chapter contains the following sections:

- Prerequisite — Database Compatibility Level (ADM)
- SQL Server Index Fragmentation Level (ADM)
- Enable Auto Detection of Robot Traffic (ADM)
- SQL Server Maintenance Plan

2.1 Prerequisite — Database Compatibility Level (ADM)

Compatibility Level effects SQL syntax and query parsing, and should have no impact of performance. Setting the Compatibility Level to a value of either SQL Server 2008(100) for servers running SQL Server 2008/2008 R2 or to SQL Server 2012(110) for servers running SQL Server 2012 take advantage of new T-SQL features, which are used in many of the scripts / commands used in the DMS Performance Tuning Guide.

2.1.1 Required Skills

- A working knowledge of SQL Server Management Studio.

2.1.2 Symptoms

- Unable to run T-SQL scripts required for DMS Tuning.

2.1.3 Checking the Database Compatibility Level

To check for the value of the database *Compatibility Level* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database and select **Properties**.
3. Select the **Options** page and look at the *Compatibility Level* property.

2.1.4 Understanding the Results

The output will look like this:

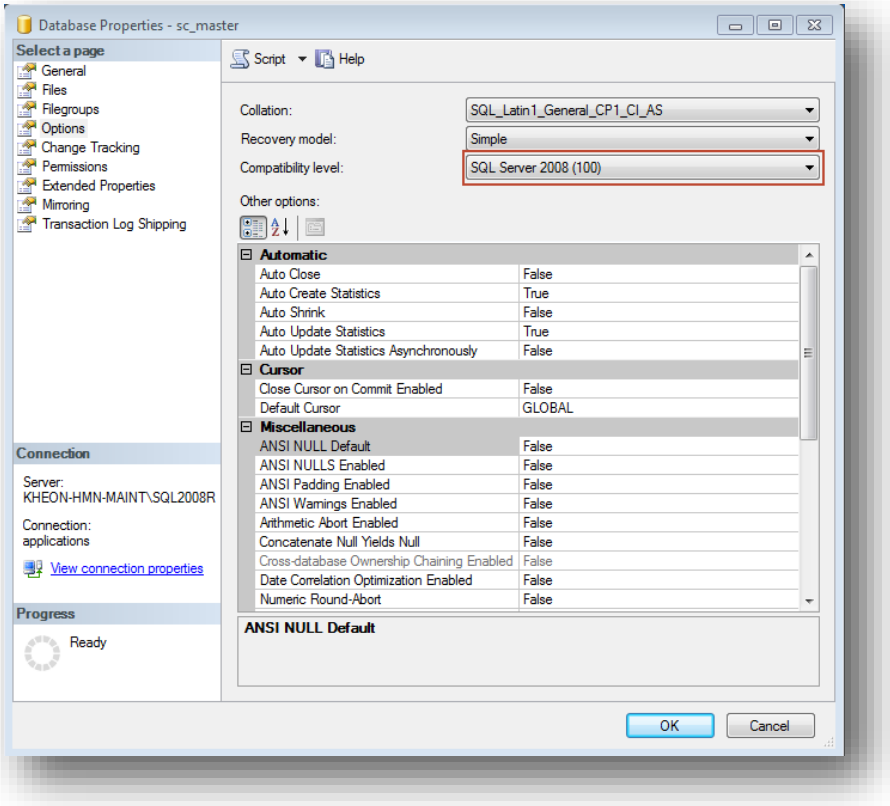


Figure 1 SQL Server 2008/2008 R2

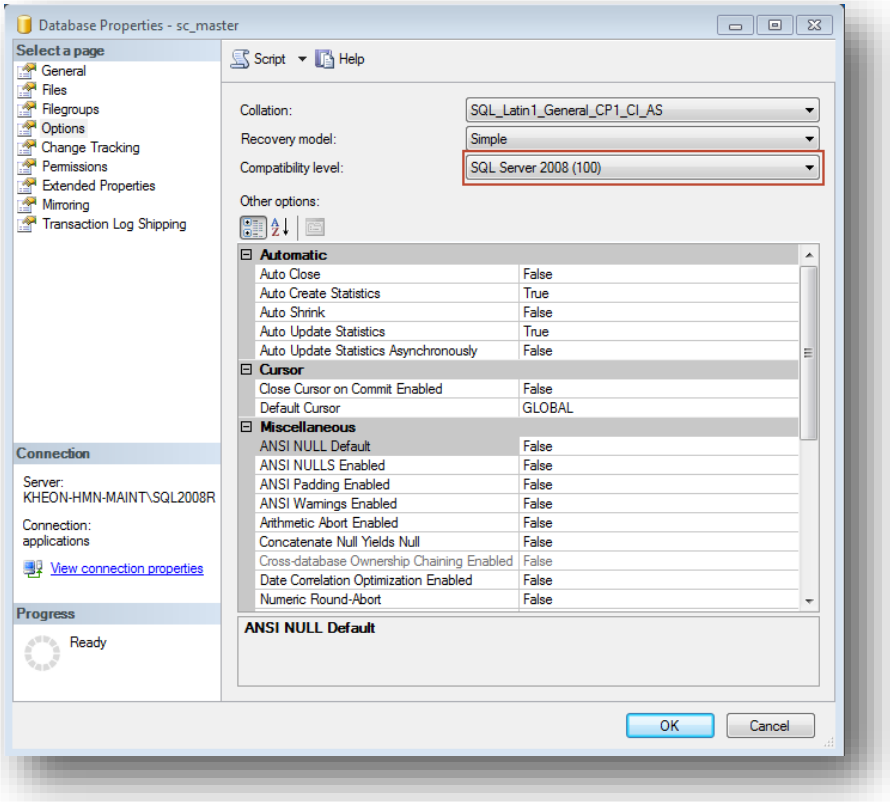


Figure 2 SQL Server 2012

Important
You must set the *Compatibility Level* property to either **SQL Server 2008(100)** when using SQL Server 2008/2008 R2 or **SQL Server 2012(110)** when using SQL Server 2012**.**

2.1.5 How to Solve

To set the *Compatibility Level* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and make sure that the *Compatibility Level* is set to either **SQL Server 2008(100)** for SQL Server 2008/2008 R2 or **SQL Server 2012(110)** for SQL Server 2012.
4. Click **OK**.

2.2 SQL Server Index Fragmentation Level (ADM)

As indexes age, insertion, and deletion of noncontiguous data can take its toll and cause fragmentation to occur. This can happen in just a few days on a busy DMS database. Minor amounts of fragmentation won't generally hurt performance. But as the percentage of fragmentation increases, performance suffers dramatically.

2.2.1 Required Skills

- Working knowledge of SQL Server Management Studio.

2.2.2 Symptoms

- Dramatic increase in CPU usage.
- Performance degradation on queries.
- Performance degradation on database writes.
- Dropped connections to the database server.
- Slow to unresponsive reports.

2.2.3 Checking for Fragmented Indexes

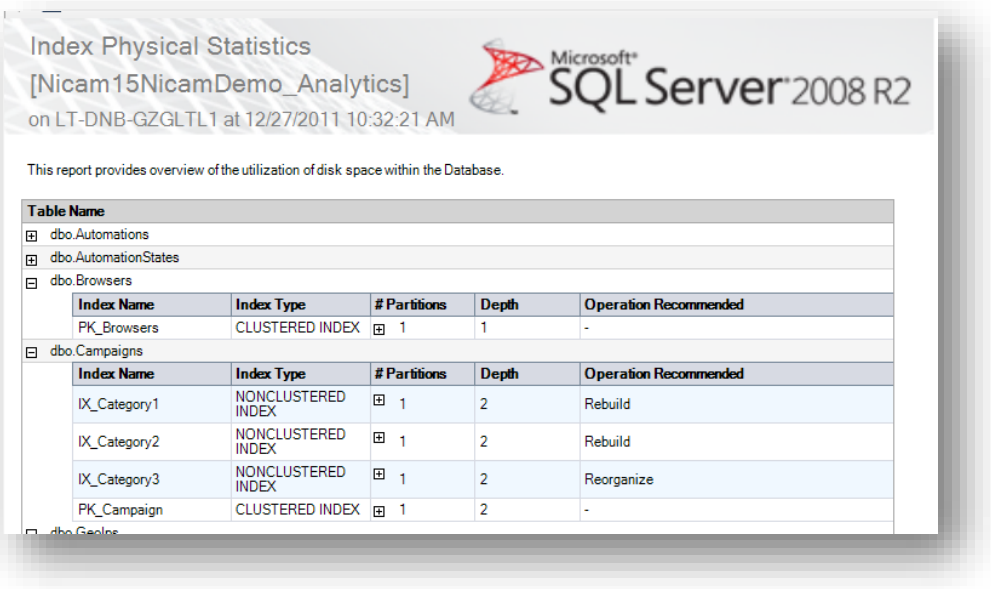
To check the percentage of fragmentation on indexes, run the Index Physical Statistics Standard Report against the DMS database:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Click **Options** page and make sure that the **Compatibility level** is set for the version of SQL Server installed.

   For more information, see the section Checking the Database Compatibility Level.
4. Click **OK**.
5. In the **Object Explorer**, right click the *DMS* database, and then click **Reports**, **Standard Reports**, **Index Physical Statistics**.
6. SQL Server Management Studio will generate a report showing information about the *Table Names, Index Names, Index Type, Number of Partitions and Operation Recommendations*.

2.2.4 Understanding the Results

The output will look like this:



Index Physical Statistics
[Nicam15NicamDemo_Analytics]
on LT-DNB-GZGLTL1 at 12/27/2011 10:32:21 AM

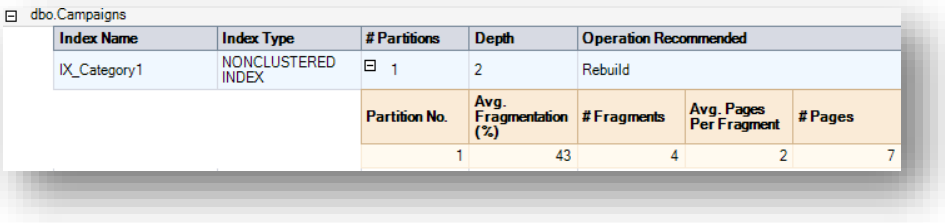
Microsoft SQL Server 2008 R2

This report provides overview of the utilization of disk space within the Database.

Table Name				
⊞ dbo.Automations				
⊞ dbo.AutomationStates				
⊟ dbo.Browsers				
Index Name	**Index Type**	**# Partitions**	**Depth**	**Operation Recommended**
PK_Browsers	CLUSTERED INDEX	⊞ 1	1	-
⊟ dbo.Campaigns				
Index Name	**Index Type**	**# Partitions**	**Depth**	**Operation Recommended**
IX_Category1	NONCLUSTERED INDEX	⊞ 1	2	Rebuild
IX_Category2	NONCLUSTERED INDEX	⊞ 1	2	Rebuild
IX_Category3	NONCLUSTERED INDEX	⊞ 1	2	Reorganize
PK_Campaign	CLUSTERED INDEX	⊞ 1	2	-

One key value that is provided in the report is the **Operation Recommended** field. If the value in this field is Rebuild, this indicates that the index is fragmented.

Expand the # **Partitions** field and you can see the % of fragmentation for a given index.



⊟ dbo.Campaigns

Index Name	**Index Type**	**# Partitions**	**Depth**	**Operation Recommended**		
IX_Category1	NONCLUSTERED INDEX	⊟ 1	2	Rebuild		
		Partition No.	**Avg. Fragmentation (%)**	**# Fragments**	**Avg. Pages Per Fragment**	**# Pages**
		1	43	4	2	7

2.2.5 Sitecore Recommendation

Sitecore recommends that you keep index fragmentation below 10%.

2.2.6 How to Solve

To defragment the indexes for the *DMS* database(s), you should execute a defragmentation maintenance plan:

1. Launch **SQL Server Management Studio.**
2. In the **Object Explorer**, expand the Management/Maintenance Plans folder.
3. Right click the *defragment indexes maintenance plan and then click **Execute**.

*If this maintenance plan does not exist, please see the section Checking for the Existence of a SQL Server Maintenance Plan.

2.2.7 Enable Auto Detection of Robot Traffic (ADM)

Robot, or web crawler traffic, occurs when search engines, such as, Google index a Web site. The amount of traffic that a web crawler can generate can be overwhelming and quickly cause the DMS databases to grow at a rapid rate.

DMS has the ability to automatically block robot traffic through configuration. This task will insure that the configuration is set to enable this feature.

2.2.8 Required Skills

- A working knowledge of Sitecore configuration files.

2.2.9 Symptoms

- Dramatic increase in DMS Analytics database size.

2.2.10 Check the Analytics.AutoDetectBots Setting

Checking that the `Analytics.AutoDetectBots` setting is enabled:

- Navigate to the `<webroot>/App_Config/Include` directory and open up the `Sitecore.Analytics.config` file in your favorite editor.

2.2.11 Understanding the Results

The output will look like this:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <settings>
      <!--  ANALYTICS AUTO DETECT BOTS
            Enable auto detection of bots.
            Default: true
      -->
      <setting name="Analytics.AutoDetectBots" value="true" />
```

The `Analytics.AutoDetectBots` parameter enables the automatic detection of robot traffic in DMS. Enabled, robot traffic is blocked before it is allowed to be captured in the database.

2.2.12 Sitecore Recommendation

For DMS, Sitecore recommends that the `Analytics.AutoDetectBots` parameter is set to *true* to block unwanted robot traffic. This will help reduce the growth of the *DMS* database.

2.2.13 How to Solve

To enable the `Analytics.AutoDetectBots` setting:

1. Navigate to the `<webroot>/App_Config/Include` directory and open up the `Sitecore.Analytics.config` file in your favorite editor.
2. Set the `Analytics.AutoDetectBots` parameter to *true*.

2.3 SQL Server Maintenance Plan

A maintenance plan eliminates the need for manual maintenance of the database(s) by running an automated set of tasks on a scheduled basis. This plan will perform regular checks and maintenance on the database(s), ensuring that the database(s) is in optimal health.

2.3.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

2.3.2 Symptoms

- Timeouts occur due to long lookup times required when the indexes become fragmented.

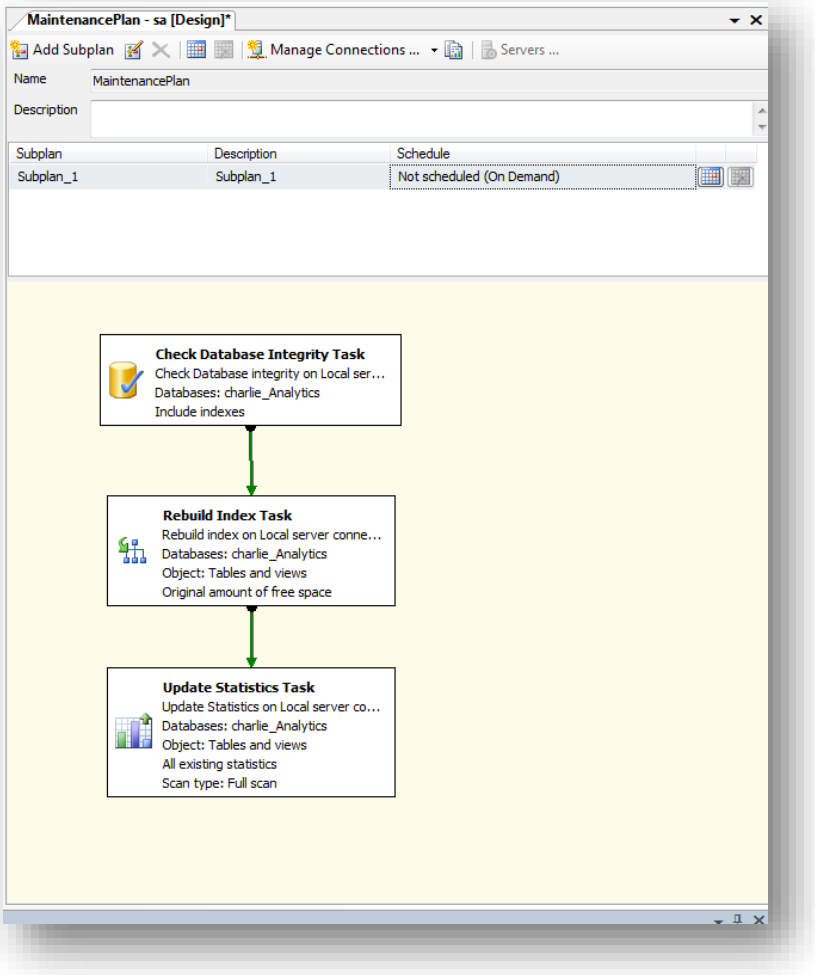
2.3.3 Checking for the Existence of a SQL Server Maintenance Plan

To check for the existence of a SQL Server Maintenance Plan, and to check that it follows Sitecore best practices:

1. Launch **SQL Server Management Studio.**
2. In the **Object Explorer**, expand the Management/Maintenance Plans folder.
3. If a maintenance plan exists, double click it to see how it is configured — this will be used in the *Findings* section of this task.

2.3.4 Understanding the Results

The output looks like this:



The maintenance plan should contain:

- A Check Database Integrity task.
- An Rebuild Index task.
- An Update Statistics task.

2.3.5 Sitecore Recommendation

Sitecore recommends that you have a SQL Server maintenance plan in place for the DMS database. The maintenance plan should contain a Check Database Integrity task, a Rebuild Index task, and an Update Statistics task.

2.3.6 How to Solve

SQL Server Management Studio contains an IDE that simplifies the creation of maintenance plans.

To create a maintenance plan for defragmenting the indexes:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, expand the Management folder.

3. Right click the *Maintenance Plans* folder and then click **New Maintenance Plan.**
4. Give the maintenance plan a meaningful name, such as, Defragment DMS Indexes**.**
5. From the **Toolbox**, drag and drop a Check Database Integrity Task, Rebuild Index Task, Update Statistics Task and place them vertically in the same order.
6. Connect the tasks together by dragging the arrow from one box to the other so they are connected as:

   Check Database Integrity Task, Rebuild Index Task, Update Statistics Task.
7. Right click the Check Database Integrity Task and then click **Edit**.
8. Select the Connection and DMS database(s) and then click **OK**.
9. Right click the Rebuild Index Task and select **Edit**.
10. Select the Connection and DMS database(s) and then click **OK**.

Note
If you are running MSSQL Server Enterprise Edition or higher, Sitecore recommends enabling *Keep indexes online while reindexing* under the Advanced Options.

11. Right click the Update Statistics Task and then click **Edit.**
12. Select Connection, DMS database(s), set the Object to Tables and Views, Update All existing statistics, Scan type = Full scan, and click **OK**.
13. Click the calendar icon next to the Schedule (upper right hand corner) and set the schedule to run daily.
14. Save your changes.

Chapter 3

Database Properties

This chapter describes some of the important database properties that you affect the performance of your DMS database.

This chapter contains the following sections:

- Recovery Model Set to Simple (ADM)
- Auto Close Property Set to False (ADM)
- Auto Shrink Property Set to False (ADM)
- Set Autogrowth Property before Inserting Data (ADM)
- Connection String Parameters

3.1 Recovery Model Set to Simple (ADM)

When you select the Simple Recovery Model setting, SQL Server logs the minimal amount of information in the transaction log. SQL Server basically truncates the transaction log whenever the transaction log becomes 70 percent full or the active portion of the transaction log exceeds the size that SQL Server could recover in the amount of time which is specified in the Recovery Interval server level configuration.

Setting the Recovery Model to Simple has the lowest amount of overhead compared Full and Bulk-logged, which is crucial to the performance requirements needed for the DMS database.

Note:
For High Availablility (HA) configurations, it is recommended that the Recovery Model be set to Full. Refer to the Sitecore Scaling Guide for more information.

3.1.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.1.2 Symptoms

- Longer times required to recover the database.

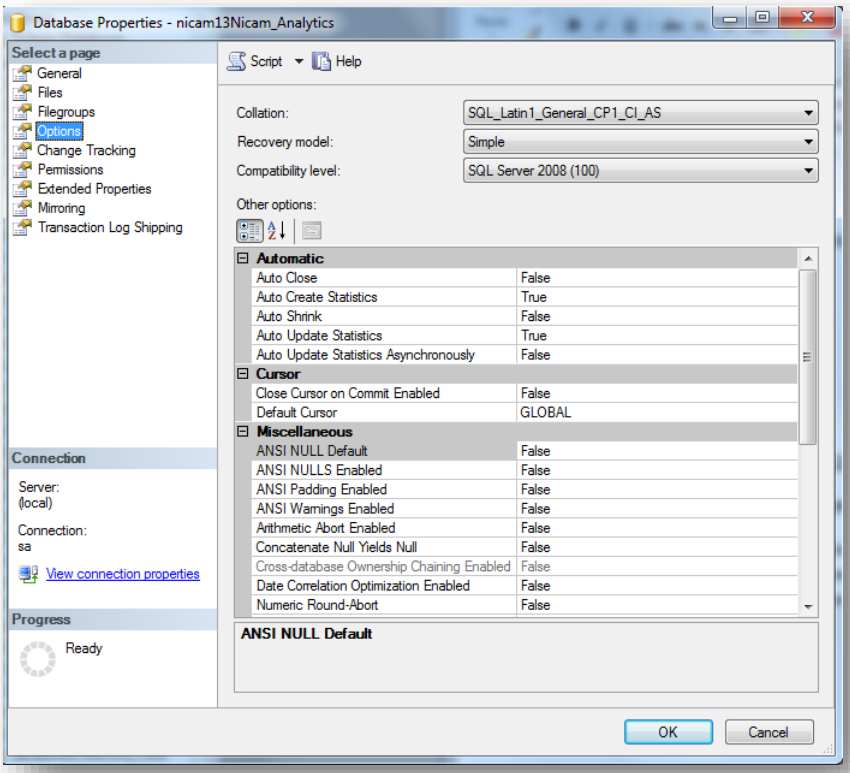
3.1.3 Checking the Recovery Model

To check for the value of the *Recovery Model* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and look at the *Recovery Model* property.

3.1.4 Understanding the Results

The output will look like this:



3.1.5 Sitecore Recommendation

Sitecore recommends that you set the *Recovery Model* property to *Simple*.

3.1.6 How to Solve

To set the *Recovery Model* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and make sure that the *Recovery Model* is set to *Simple*.

3.2 Auto Close Property Set to False (ADM)

When SQL Server opens a database, resources are allocated to maintain that state. Memory for locks, buffers, security tokens, and so on, are all assigned.

These operations take time. The *Auto Close* property dictates how these resources are handled. If it is set to *true* or *ON*, then when the last connection is closed these resources are deallocated. This may seem like a good thing, but if a new connection comes in within a short period of time (1/10th of second or quicker), then all of those resources need to be started again. Setting the *Auto Close* property to *false* or *OFF* will prevent this from happening.

3.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.2.2 Symptoms

- Longer times required to connect to the database.

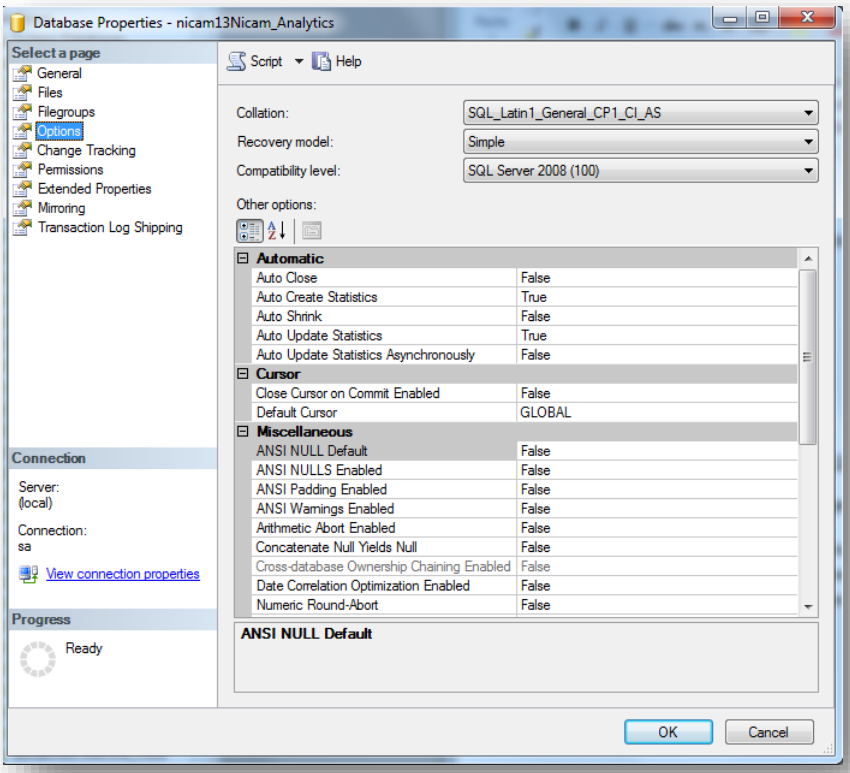
3.2.3 Checking the Value of the Auto Close Property

To check for the value of the *Auto Close* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and look at the *Auto Close* property.

3.2.4 Understanding the Results

The output will look like this:



3.2.5 Sitecore Recommendation

Sitecore recommends that you set the *Auto Close* property to *False*.

3.2.6 How to Solve

To set the *Auto Close* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and make sure that the *Auto Close* property is set to *False*.

3.3 Auto Shrink Property Set to False (ADM)

The *Auto Shrink* property has some downsides:

- It uses a lot of resources when it's called.
- You cannot control when it is called.

If you combine *Auto Shrink* with *Auto Growth*, you can get into a spiral of constantly growing and shrinking the database, taking valuable resources away from other database tasks as well as causing fragmentation issues. If a database or file needs to be shrunk, it should be done so with a script, command or a scheduled Maintenance Plan. Setting the *Auto Shrink* property to *false* or *OFF* will disable this feature.

3.3.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.3.2 Symptoms

- Performance degradation.

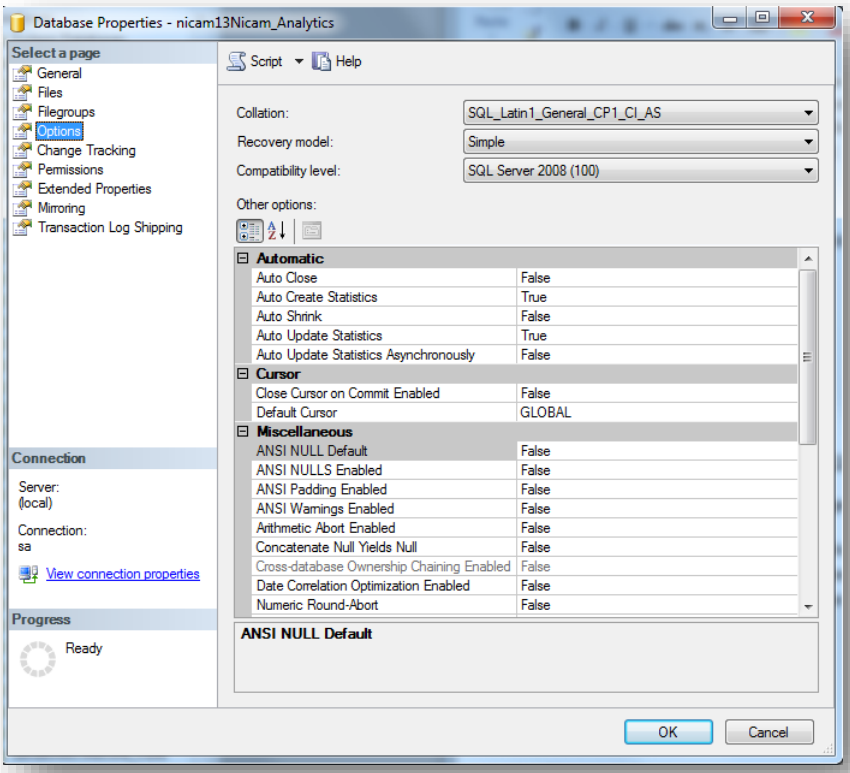
3.3.3 Checking the Value of the Auto Shrink Property

To check for the value of the *Auto Shrink* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Options** page and look at the *Auto Shrink* property.

3.3.4 Understanding Results

The output will look like this:



3.3.5 Sitecore Recommendation

Sitecore recommends that the *Auto Shrink* property be set to *False*.

3.3.6 How to Solve

To set the *Auto Shrink* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.

Select the *Options* page and make sure that the *Auto Shrink* property is set to *False*.

3.4 Set Initial Size Value before Inserting Data

It's important to set the *Initial Size* value of the database to a value which will accommodate 3 – 6 months worth of data. This will reduce the frequency with which *Autogrowth* occurs.

The *Autogrowth* operation is not only expensive in terms of resources required to perform the operation, but also causes fragmentation issues.

3.4.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.4.2 Symptoms

- Performance degradation.
- Excessive resource consumption due to frequent Autogrowth commands.
- Excessive index fragmentation.

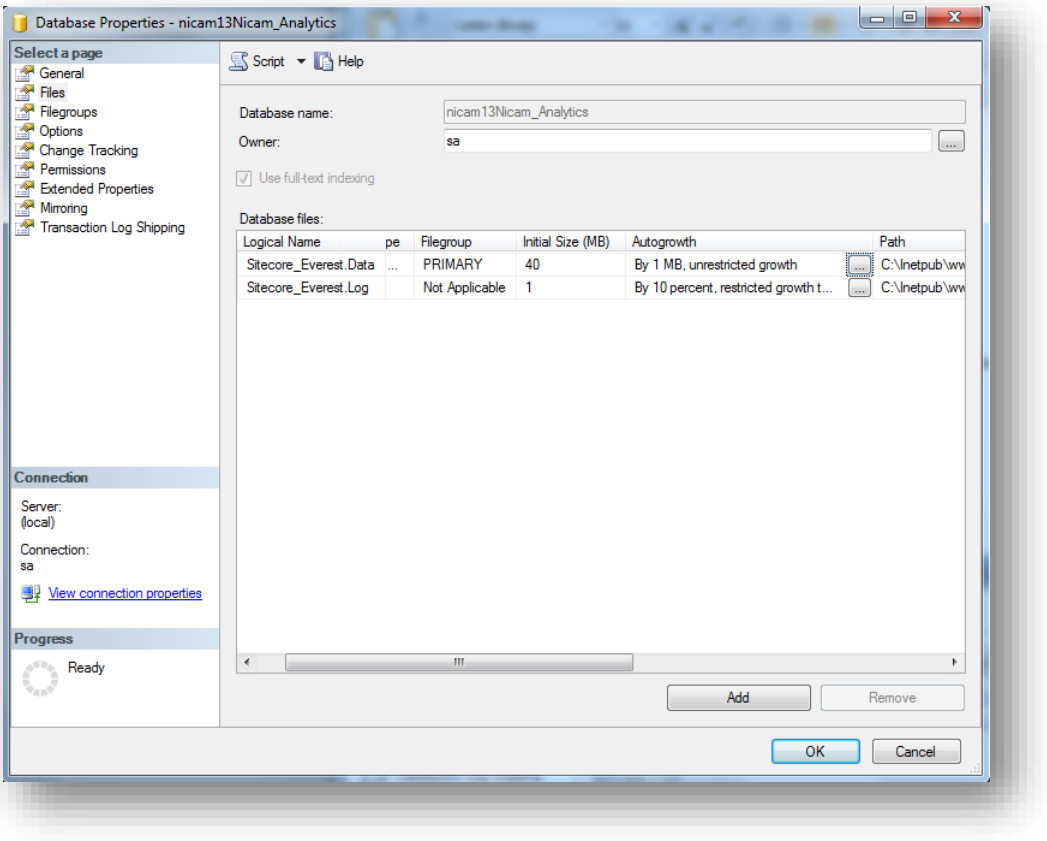
3.4.3 Checking the Initial Size Value

To check the *Initial Size* value:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Files** page and look at the *Initial Size* value.

3.4.4 Understanding the Results

The output will look like this:



3.4.5 Sitecore Recommendation

Sitecore recommends that you set the *Initial Size* value to accommodate 3 – 6 months worth of data storage.

3.4.6 How to Solve

To set the *Initial Size* value:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Files** page and set the *Initial Size* value to 3 – 6 months worth of data storage.

3.5 Set Autogrowth Property before Inserting Data (ADM)

Setting the *Autogrowth* property incorrectly can have performance implications. If the *Autogrowth* property is set too low and a transaction requires more space, this transaction will have to wait for the growth to occur before it can be completed. Furthermore, growing the database too frequently will cause fragmentation issues.

The exact value to use in your configuration setting and whether to choose between a percentage growth and a specific MB size growth depends on many factors in your environment. A general rule of thumb that you can use for testing is to set the Autogrowth setting to about 1/8th of the size of the file.

3.5.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

3.5.2 Symptoms

- Performance degradation.

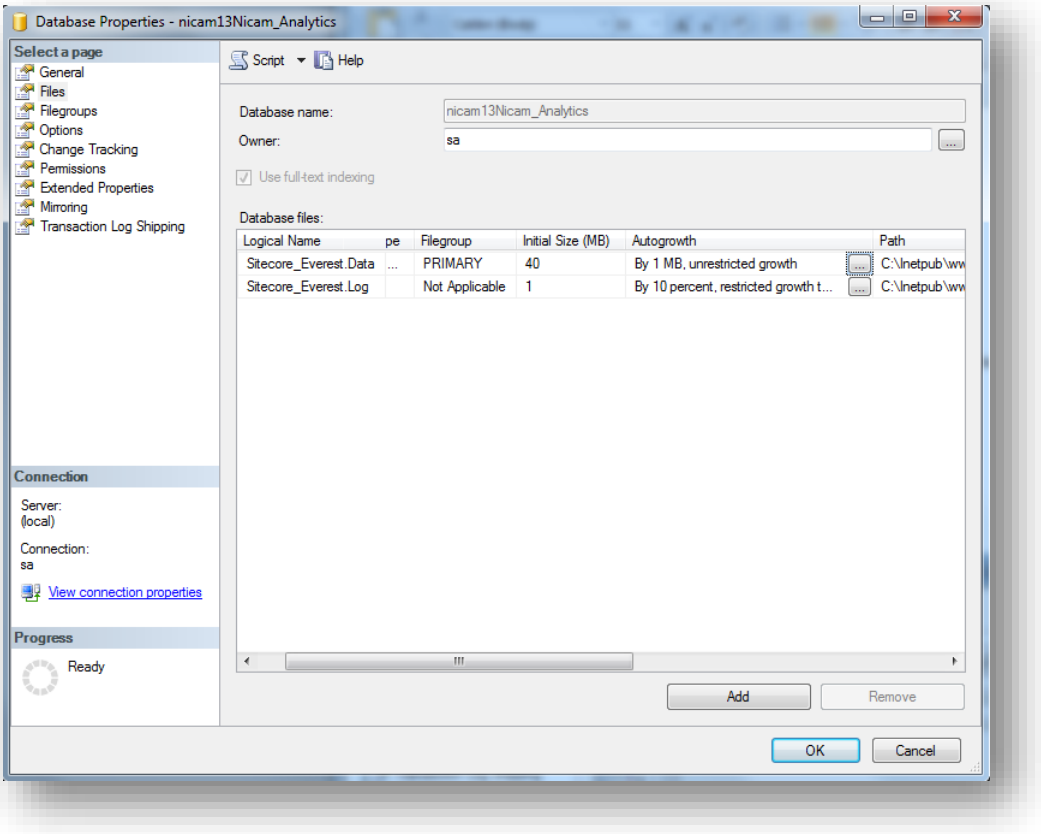
3.5.3 Checking the Value of the Autogrowth Property

To check for the value of the *Autogrowth* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the **Files** page and look at the *Autogrowth* property.

3.5.4 Understanding the Results

The output will look like this:



3.5.5 Sitecore's Recommendation

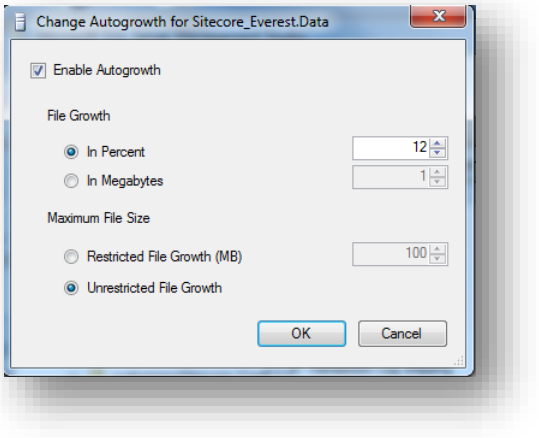
Sitecore recommends that you enable the *Autogrowth* property and set the growth rate to 10 – 15%.

3.5.6 How to Solve

To enable and set the *Autogrowth* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the *DMS* database, and then click **Properties**.
3. Select the Files page and in the *Autogrowth* property, click the … button to launch the **Change Autogrowth** dialog box.
4. Select the **Enable Autogrowth** check box.
5. In the **File Growth** section, select the **In Percent** option and set the value to 10 to 15% (approx. 1/8th the size of the database).
6. In the **Maximum File Size** section, select the **Unrestricted File Growth(MB)** option.

The dialog box should look like this:



3.5.7 Notes and Comments

If the site is experiencing, or if you expect it to experience high traffic loads, the initial amount of space allocated to the *DMS* database should be set as large as possible. This will help reduce the frequency with which autogrowth occurs.

3.6 Connection String Parameters

The analytics connection string, found in the `<webroot>/App_Config/ConnectionString.config`, has parameters that control how the connection string is maintained as well as when the connection string times out. Setting these correctly will aid in the performance of communication between Sitecore CMS and the Sitecore DMS Analytics database.

The two connection string parameters that should be checked are:

- Connection Timeout
  - The Connection Timeout should only be set if there are TCP errors in the log files related to connecting to the analytics database.
- Min Pool Size

3.6.1 Required Skills

- A working knowledge of Sitecore configuration files.
- A working knowledge on analyzing Sitecore log files.

3.6.2 Symptoms

- TCP timeout errors.
- Creating a connection to the DMS database server takes an excessive amount of time.

3.6.3 Checking the Analytics Connection String Parameters:

To check the parameters of the Analytics Connection String:

1. Navigate to the `<webroot>/App_Config` directory and open up the `ConnectionString.config` file in your favorite editor.

3.6.4 Understanding the Results

The output will look like this:

```
        <?xml version="1.0" encoding="utf-8"?>
        <connectionStrings>
          <!--
            Sitecore connection strings.
            All database connections for Sitecore are configured here.
          -->
          <add name="core" connectionString="user id=sa;password=xxxx;Data
Source=(local);Database=CMS Core"/>
          <add name="master" connectionString="user id=sa;password=xxxx;Data
Source=(local);Database=CMS Master"/>
          <add name="web" connectionString="user id=sa;password=xxxx;Data
Source=(local);Database=CMS_Web"/>
          <add name="analytics"
            connectionString="user id=sa;password=xxxx;Data
Source=(local);Database=DMS_Analytics;Connection Timeout=120;Min Pool Size=5"/>
        </connectionStrings>
```

The `Connection Timeout` parameter is the length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error. The default value is 15 — seconds.

The `Min Pool Size` parameter displays the minimum number of connections allowed in the pool. The default value is 0, meaning that when all connections are closed the pool is empty. A value of > 0 will insure that a connection is always available from the pool, rather than have re-instantiate a new connection each time all connections are closed.

3.6.5 Sitecore Recommendation

Sitecore recommends setting the following Connection String Parameter as such:

- Connection Timeout=120 (only if TCP errors appear in the log files, otherwise the `Connection Timeout` parameter can be removed, using it's default of 15 seconds)
- Min Pool Size=5

3.6.6 How to Solve

To set the `Connection String` parameters:

1. Navigate to the `<webroot>/App_Config` directory and open up the `ConnectionString.config` file in your favorite editor.
2. Edit the analytics connection string and save the file.

Chapter 4

Server Recommendations

When sizing hardware for your DMS implementation the high volume, high traffic nature of a website needs to be taken into consideration. The server recommendations provided are focused on an enterprise level deployment of SQL Server.

This chapter contains the following sections:

- Server Recommendations for DMS

4.1 Server Recommendations for DMS

This check is used to determine if the server configuration that you use to host the DMS Analytics database is sufficient for optimal performance. Starting with a minimum baseline, a scoring system is used as a means to record the performance level of the server. The scoring is broken down into sections, with each section having its own value. A score of 0 in any section indicates that this section meets the minimum Sitecore recommendation. A score > 0 indicates that the database should be able to perform at a higher level. And a score of < 0 is a red flag, indicates that additional resources, or changes, need to be made to bring the server up to at least the Sitecore recommendations.

The scoring system is based on the minimum requirements listed in the DMS Server Requirements on the SDN:

Sitecore recommends the following minimum server requirements for the DMS Analytics database:

- Dedicated physical server
  - Minimum of one quad core CPU
  - Minimum of 8GB RAM
  - 1 disk for OS
  - 2 disks for database (1 for database, 1 for logs)
  - 1 disk for TempDb
  - Disks setup as RAID according to MS SQL Server Requirements
- Windows Server 2003, 2008, or 2008 R2 (highly recommended)
  - 64 bit version recommended
- Microsoft SQL Server 2008 (2008 R2 recommended) or 2012.
  - 64 bit version recommended

4.1.1 Required Skills

- A working knowledge of system infrastructures.

4.1.2 Symptoms

- Sustained high CPU loads.
- Excessive memory consumption.
- Excessive I/O wait times and disk usage.
- Performance degradation.
- Report timeouts.

4.1.3 Scoring Server Configuration

Scoring the server configuration is divided into two general categories, with each category broken down into scoring tables.

The score recorded in each table determines whether or not the system meets, exceeds, or falls short of the Sitecore recommendations.

General Requirements / Installed Software

SQL Server version	2008 / 2012	2005
	+1	-1

Score _____

Is SQL Server the only application running on the server?	Yes	No
	+1	-1

Score _____

Is DMS the only database on the server?	Yes	No
	+1	0

Score_____

Hardware Requirements

RAM	Less than 8 GB	8GB	16GB	24GB	32GB +
Standalone Server	-1	0	+1	+2	+3
Virtual Server	-2	-1	0	+1	+2

Score_____

CPU	1 core	2 core	4 core	8 core	16 core +
Standalone Server	-1	0	+1	+2	+3
Virtual Server	-2	-1	0	+ 1	+2

Score_____

Separate Data Disk	Yes	No
	+1	-1

Score_____

Separate Log Disk	Yes	No
	+1	-1

Score_____

Separate TempDb Disk	Yes	No
	+1	-1

Score_____

Chapter 5

Analytics Configuration Parameters

This chapter focuses on two parameters, located in the `Sitecore.Analytics.config` file, that can have an impact on how frequently information is flushed from memory and written to the database.

These two parameters are `Analytics.TrackerChanges.FlushInterval` and `Analytics.TrackerChanges.MaxRows`.

This chapter includes the following sections:

- Parameter Behavior

5.1 Parameter Behavior

5.1.1 Analytics.TrackerChanges.FlushInterval

This parameter controls the following behavior:

- From one side: max time data is stored in memory before it gets written to the database.
- From the other side: frequency of requests to the SQL Server (matters for the low-latency servers).
- Smaller values increase frequency of requests to SQL server => bigger load on the SQL Server.
- Bigger values increase the risk of losing data in case of server crash/restart (when data queue is not overflown, and the flushes occur at intervals).

5.1.2 Analytics.TrackerChanges.MaxRows

This parameter controls the following behavior:

- From one side: max memory consumption by the commit queue (indirectly).
- From the other side: max amount of data that will be lost in case of server crash.
- Smaller values increase frequency of requests to SQL server due to queue being overloaded => bigger load on the SQL Server.
- Bigger values increase memory consumption and amount of data lost in case of server crash/restart.

5.1.3 Relationship Between FlushInterval and MaxRows

Approximate equivalence relation between FlushInterval and MaxRows is:

MaxRows ~= RWS * FlushInterval / 1000 (1)

Where RWS is [Rows per Second], database performance measure and has an estimation of:

- RWS = RQS * RPR, where
- RQS is number of tracked website requests per second (planned or measured by IIS/ASP.NET performance counters)
- RPR is average number of DB rows written per request:
  - RPR = 3 + PR + PE + A, where
  - 3 rows always written are: current page, previous page, current visit
  - PR is the average number of different profiles defined on a content page
  - PE is the average number of Goals and Page Events defined on a content page
  - A is the number of engagement automation plans that an average visitor will be in.

When (1) is fulfilled, commit queue will accumulate MaxRows records approximately at the same time as FlushInterval runs out, resulting in the settings having the same effect on the system.

The parameters that can be tuned:

- (Stress on SQL Server and effect of SQL Server latency) vs. (Delay when writing data and Memory consumption)

5.2 Tuning Scenarios

5.2.1 SQL Server handles the traffic, SQL Server latency is low:

Strategy is to minimize data writing delay at the cost of putting more stress on SQL Server (e.g. “flush not later than every 1-10 seconds”):

- FlushInterval = 1000 - 10000
- MaxRows < RWS * FlushInterval / 1000

5.2.2 SQL Server can handle the traffic, but SQL Server latency is high:

Strategy is to minimize the effect of SQL Server latency by flushing with bigger intervals — for example, flush not faster than every (10-100) SQL Server ping latency values:

- FlushInterval = (100-1000) * SQL Server latency
  - For example, let SQL Server latency be 100ms, FlushInterval = 10000 - 100000
- MaxRows > RWS * FlushInterval / 1000
  - Top limit for MaxRows is memory consumption.
  - For example, with RWS ~= 100
    - MaxRows > 10000

5.2.3 SQL Server cannot handle the traffic:

Strategy is to reduce the number of tracked requests with help of Analytics.Sampling.Percentage setting, until SQL Server can handle the load.