



# OMS 1 for Sitecore CMS 6.1 – 6.4

# OMS Performance Tuning Guide

A developer's guide to optimizing the performance of Sitecore OMS

The information contained in this document represents the current view of Sitecore Corporation on the issues discussed as of the date of publication and is subject to change at any time without notice. This document and its contents are provided AS IS without warranty of any kind, and should not be interpreted as an offer or commitment on the part of Sitecore, and Sitecore cannot guarantee the accuracy of any information presented. **SITECORE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.**

The descriptions of other companies' products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Sitecore. Sitecore cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

All trademarks are the property of their respective companies

©2012 Sitecore Corporation. All rights reserved.

## Table of Contents

Chapter 1	Introduction .....	4
1.1	SQL Server Index Fragmentation Level .....	5
1.2	Removing Robot Traffic.....	7
1.3	Exclude Robot Traffic from the OMS Analytics Database.....	10
1.4	SQL Server Maintenance Plan .....	11
1.5	Check GlobalSession.CookieValue DB Storage Length .....	14
1.6	Adding/Modifying Indexes On The GlobalSessions And MVTestVariablePage Tables.....	17
Chapter 2	Database Properties.....	20
2.1	Compatibility Level Set to SQL Server 2008 (100) .....	21
2.2	Recovery Model Set to Simple.....	23
2.3	Auto Close Property Set to False.....	25
2.4	Auto Shrink Property Set to False.....	27
2.5	Set Initial Size Value before Inserting Data .....	29
2.6	Set Autogrowth Property before Inserting Data .....	31
2.7	Server Requirements for OMS Database.....	34
2.8	OMS Report Timeout Settings.....	37
2.9	Connection String Parameters.....	38

# Chapter 1

## Introduction

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

Each task contain introductory information, symptoms that it is likely to solve, a series of steps to check, how to understand the results, Sitecore recommendations, how to solve the issue, results to record, and go no-go results based on what has been checked.

This chapter contains the following sections:

- SQL Server Index Fragmentation Level
- Removing Robot Traffic
- Exclude Robot Traffic from the OMS Analytics Database
- SQL Server Maintenance Plan

## 1.1 SQL Server Index Fragmentation Level

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 OMS database. Minor amounts of fragmentation won't generally hurt performance. But as the percentage of fragmentation increases, performance suffers *dramatically*.

### 1.1.1 Required Skills

- Working knowledge of SQL Server 2008 Management Studio

### 1.1.2 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.

### 1.1.3 Checking for Fragmented Indexes

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

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the OMS database, and then click **Properties**.
3. In the **Database Properties** window, select the *Options* page and make sure that the **Compatibility level** is set to *SQL Server 2008 (100)*.
4. In the **Object Explorer**, right click the OMS database, and then click **Reports, Standard Reports, Index Physical Statistics**.
5. SQL Server Management Studio will generate a report showing information about the *Table Names, Index Names, Index Type, Number of Partitions and Operation Recommendations*.

### 1.1.4 Understanding the Results

The output will look like this:

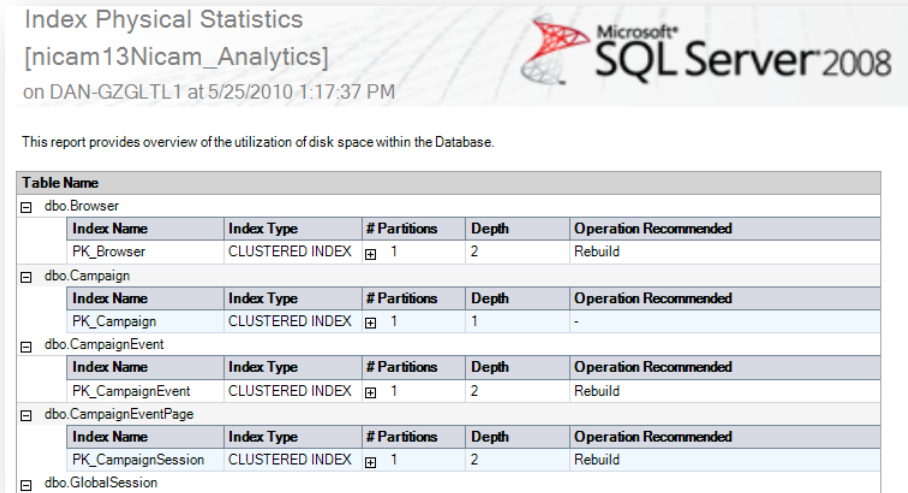


Table Name	Index Name	Index Type	# Partitions	Depth	Operation Recommended
dbo.Browser	PK_Browser	CLUSTERED INDEX	1	2	Rebuild
dbo.Campaign	PK_Campaign	CLUSTERED INDEX	1	1	-
dbo.CampaignEvent	PK_CampaignEvent	CLUSTERED INDEX	1	2	Rebuild
dbo.CampaignEventPage	PK_CampaignSession	CLUSTERED INDEX	1	2	Rebuild
dbo.GlobalSession					

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.

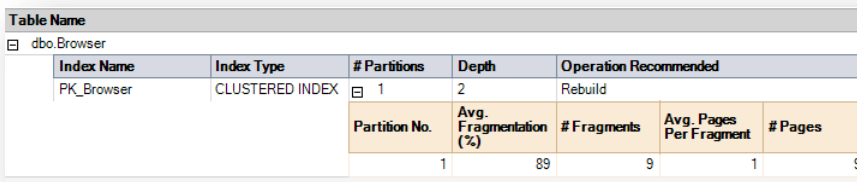


Table Name	Index Name	Index Type	# Partitions	Depth	Operation Recommended		
dbo.Browser	PK_Browser	CLUSTERED INDEX	1	2	Rebuild		
			Partition No.	Avg. Fragmentation (%)	# Fragments	Avg. Pages Per Fragment	# Pages
			1	89	9	1	9

### 1.1.5 Sitecore Recommendation

Sitecore recommends that you keep index fragmentation below 10%.

### 1.1.6 How to Solve

To defragment the indexes for the OMS 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 refer to the *Checking for the Existence of a SQL Server Maintenance Plan* task.

## 1.2 Removing Robot Traffic

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 OMS databases to grow at a rapid rate.

The *Robot Removal* script removes unwanted information from the OMS database, while keeping your user specific data intact.

While the *Robot Removal* script removes unwanted information from the OMS database, it does not reduce the amount of disk space required. Running the *Shrink* database task will to this for you.

### 1.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio
- A working knowledge of how to run SQL scripts.

### 1.2.2 Symptoms

- Dramatic increase in OMS Analytics database size.

### 1.2.3 Checking if the Robot Removal Script is Run at Regular Intervals:

1. Checking to see if the *Remove Robots* script (*Remove BOTs CleanDB.sql*) script is being used is just a matter of checking to see if the db admin is running the script, followed by the *Shrink* database task.
2. Run the following script to check if robot traffic exists in the OMS database:

```
--Find robot records
USE [***OMS***]

SELECT TOP (50) [IpAddress],COUNT (IpAddress) AS 'Total count'
FROM [Session]
GROUP BY [IpAddress]
ORDER BY [Total count] DESC
GO
```

### 1.2.4 Understanding the Results

The output looks like this:

IpAddress	Total count
<b>66.249.72.178</b>	502116
<b>213.150.46.211</b>	423920
<b>66.249.65.51</b>	258894

Look for internal IP addresses that have generated a large number of records.

## 1.2.5 Sitecore Recommendation

For OMS, Sitecore recommends that you run the *Robot Removal* script, along with the *Shrink* database task at a scheduled interval, for example, daily.

## 1.2.6 How to Solve

To run the `Remove BOTs CleanDB.sql` script:

1. Launch **Microsoft SQL Server Management Studio**.
2. In the **Object Explorer**, expand the *Databases* folder.
3. Select the OMS database.
4. Click **File, Open File** to open the `Remove BOTs CleanDB.sql` script  
Alternatively, right click the OMS database and then click **New Query**.  
Cut and Paste the `Remove BOTs Clean DB.sql` script into the query window.
5. Execute the script.

To run the *Shrink* database task:

1. Right click the OMS database and the click **Tasks, Shrink, Database**.
2. Run the task.

## 1.2.7 Notes and Comments

The `Remove BOTs CleanDB.sql` script:

```

--- Main idea of the script is to remove all records concerning bots from the
database.
--- The bot is a global session with 'VisitorIdentification' property value > 900
--- The data is cleared starting from the tables that refers 'Page' and 'Session'
--- tables.
-- Remember IDs of bots' global sessions
create table #BotGlobalSessionIDs(
    globalSessionID uniqueidentifier not null
)
insert into #BotGlobalSessionIDs (globalSessionID)
select GlobalSessionId from [dbo].[GlobalSession] where VisitorIdentification > 900
-- Clearing Tags for bot global sessions
delete from [dbo].[Tag]
where GlobalSessionId in
(
    select globalSessionId from #BotGlobalSessionIDs
)
--remembering bot pages
create table #BotPageIDs(
    pageID uniqueidentifier not null
)
insert into #BotPageIDs
select PageId from [dbo].Page
where SessionId in (
    select SessionId from [dbo].[Session]
    where GlobalSessionId in
        (
            select globalSessionID from #BotGlobalSessionIDs
        )
)
--remove data from 'MVTTestVariablePage'
delete from [dbo].MVTTestVariablePage
where PageId in (
    select PageId from #BotPageIDs
)

delete from CampaignEventPage
where PageId in

```



```
(
    select PageId from #BotPageIDs
)

delete from PageEvent
where PageId in
(
    select PageId from #BotPageIDs
)

delete from [dbo].Page
where PageId in
(
    select PageId from #BotPageIDs
)

drop table #BotPageIDs

-- Start clearing bot sessions
create table #BotSessionIDs (
    sessionID uniqueidentifier not null
)
insert into #BotSessionIDs
select SessionId from [dbo].[Session]
where GlobalSessionId in
(
    select GlobalSessionId from #BotGlobalSessionIDs
)

delete from [dbo].ProfileKey
where ProfileId in
(
    select ProfileId from [dbo].[Profile]
    where SessionId in
    (
        select SessionId from #BotSessionIDs
    )
)

delete from [dbo].[Profile]
where SessionId in
(
    select SessionId from #BotSessionIDs
)

delete from [dbo].[Session]
where SessionId in
(
    select SessionId from #BotSessionIDs
)

drop table #BotSessionIDs

--Clear bots' global sessions
delete from [dbo].[GlobalSession] where GlobalSessionId in (
    select GlobalSessionId from #BotGlobalSessionIDs
)
drop table #BotGlobalSessionIDs
```

## 1.3 Exclude Robot Traffic from the OMS Analytics Database

Exclude robot traffic is a feature that was introduced in Sitecore 6.2.0 rev. 100507, OMS 1.1.1 rev. 100507 (6.2 update 2) and is enabled by default. However, this is not available in older versions.

The *Exclude Robot Traffic* task is used to determine if updating is required.

Robot traffic from internal search appliances, such as Google, can add unwanted session records to the OMS database. This can rapidly cause extensive unwanted database growth.

### 1.3.1 Required Skills

- A working knowledge of checking the installed version of Sitecore.

### 1.3.2 Symptoms

- Dramatic increase in the size of the OMS Analytics database.

### 1.3.3 Checking the Sitecore Version

To check the Sitecore version:

1. Browse to the Sitecore log in page. For example: <http://mysite/sitecore>
2. The Sitecore version number and revision are located in the upper right hand corner of the screen.

### 1.3.4 Understanding the Results

If the Sitecore version is 6.2.0 rev. 100507 or higher, the *Exclude Robot Traffic* task is available.

If the version of Sitecore is less than 6.2.0 rev. 100507, the *Exclude Robot Traffic* task is not available and an upgrade may be required.

### 1.3.5 Sitecore Recommendation

Sitecore recommends upgrading to Sitecore 6.2.0 rev. 100507 or later, along with upgrading OMS to version 1.1.1 rev. 100507 or later to take advantage of the *Exclude Robot Traffic* feature.

In cases where an upgrade is not possible, `Remove BOTs CleanDB.sql` script should be setup as a maintenance plan and run at a scheduled interval. For more information about the script, see the *Removing Robot Traffic* section.

### 1.3.6 How to Solve

Upgrade Sitecore and OMS to Sitecore 6.2.0 rev. 100507 and OMS 1.1.1 rev. 100507 by following the upgrade procedures available on the SDN at:

[Sitecore CMS 6 Updates](#)

[Sitecore OMS Updates](#)

## 1.4 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.

### 1.4.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 1.4.2 Symptoms

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

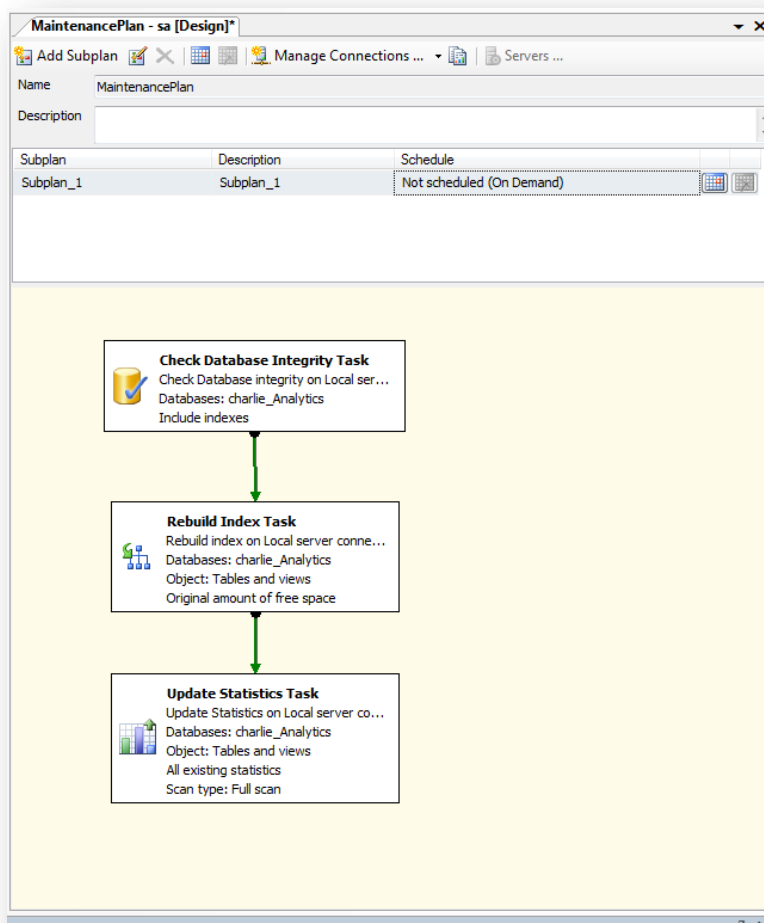
### 1.4.3 Checking for the Existence of a SQL Server Maintenance Plan

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.

## 1.4.4 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.

## 1.4.5 Sitecore Recommendation

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

## 1.4.6 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 OMS 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.

## 1.5 Check GlobalSession.CookieValue DB Storage Length

The default field type, and length, for the CookieValue in the GlobalSession table is a nvarchar(MAX). This means that it has huge storage capabilities and cannot be indexed. Even though the use of nvarchar(MAX) is designed to hold a large GUID value, there are performance implications in doing so.

A GUID will always be fixed at 32 characters, so this task is designed to change the CookieValue field of the GlobalSession table to type varchar(32).

### 1.5.1 Required Skills

- A working knowledge of running T-SQL scripts.

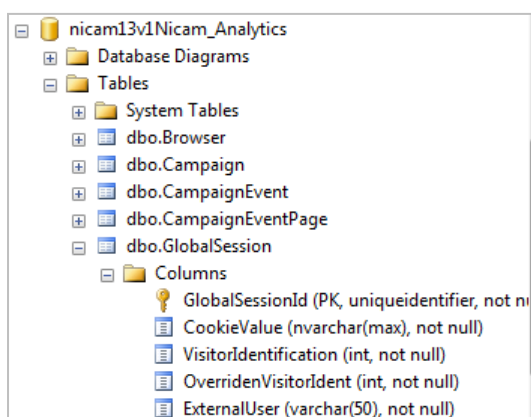
### 1.5.2 Symptoms

- Performance issues due to not being able to index the CookieValue field.

### 1.5.3 How to check if GlobalSession.CookieValue field type is nvarchar(MAX)

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, expand the *OMS database / Tables / GlobalSession / Columns* node.

### 1.5.4 Understanding the Results



The field type and size for the CookieValue is shown as nvarchar(MAX).

### 1.5.5 Sitecore Recommendation

Sitecore recommends that the GlobalSession.CookieValue field type and size be set to varchar(32), and there should be a non clustered index on this field to benefit from increased performance.

### 1.5.6 How to Query for any Cookie Values that are > 32 Characters

Prior to running the script to change the CookieValue field from nvarchar(MAX) to varchar(32), you need to check for the existence of any possible cookie values that are > 32 characters. If any are found, they need to be removed from the table(s).

1. Launch **SQL Server Management Studio**.

2. In the **Object Explorer**, right click on the OMS database and select **New Query**.
3. Execute the following script:

```
SELECT GlobalSession.GlobalSessionId FROM GlobalSession
WHERE (LEN(GlobalSession.CookieValue) > 32)
```

If there are any GlobalSession.CookieValue fields that have > 32 characters, they will appear in the results. If there are values shown, please go to the *To remove any CookieValue that contains > 32 characters* step, otherwise go to the *To change CookieValue field from nvarchar(MAX) to varchar(32)* step.

To remove any CookieValue that contains > 32 characters:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click on the OMS database and select **New Query**.
3. Execute the following script:

```
USE <your db name here>
GO

CREATE TABLE #GlobalSessionIDs
(
    globalSessionID UNIQUEIDENTIFIER NOT NULL
)

INSERT INTO #GlobalSessionIDs (globalSessionID)
SELECT GlobalSessionId
FROM [dbo].[GlobalSession]
WHERE LEN(GlobalSession.CookieValue) > 32

-- Clearing Tags for global sessions
DELETE FROM [dbo].[Tag]
WHERE GlobalSessionId IN
(
    SELECT globalSessionId FROM #GlobalSessionIDs
)

--remembering pages
CREATE TABLE #PageIDs
(
    pageID UNIQUEIDENTIFIER not null
)

INSERT INTO #PageIDs
SELECT PageId FROM [dbo].Page
WHERE SessionId IN
(
    SELECT SessionId FROM [dbo].[Session]
    WHERE GlobalSessionId IN
    (
        SELECT globalSessionID FROM #GlobalSessionIDs
    )
)

--remove data from 'MVTestVariablePage'
DELETE FROM [dbo].MVTestVariablePage
WHERE PageId IN
(
    SELECT PageId FROM #PageIDs
)

DELETE FROM CampaignEventPage
WHERE PageId IN
(
    SELECT PageId FROM #PageIDs
)

DELETE FROM PageEvent
WHERE PageId IN
(
    SELECT PageId FROM #PageIDs
```

```
)

DELETE FROM [dbo].Page
WHERE PageId IN
(
    SELECT PageId FROM #PageIDs
)

DROP TABLE #PageIDs

-- Start clearing sessions
CREATE TABLE #SessionIDs
(
    sessionID UNIQUEIDENTIFIER not null
)

INSERT INTO #SessionIDs
SELECT SessionId FROM [dbo].[Session]
WHERE GlobalSessionId IN
(
    SELECT GlobalSessionId FROM #GlobalSessionIDs
)

DELETE FROM [dbo].ProfileKey
WHERE ProfileId IN
(
    SELECT ProfileId FROM [dbo].[Profile]
    WHERE SessionId IN
    (
        SELECT SessionId FROM #SessionIDs
    )
)

DELETE FROM [dbo].[Profile]
WHERE SessionId IN
(
    SELECT SessionId FROM #SessionIDs
)

DELETE FROM [dbo].[Session]
WHERE SessionId IN
(
    SELECT SessionId FROM #SessionIDs
)

DROP TABLE #SessionIDs

--Clear global sessions
DELETE FROM [dbo].[GlobalSession] WHERE GlobalSessionId IN
(
    SELECT GlobalSessionId FROM #GlobalSessionIDs
)

DROP TABLE #GlobalSessionIDs
```

To change CookieValue field from nvarchar(MAX) to varchar(32):

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click on the OMS database and select **New Query**.
3. Execute the following scripts:

```
ALTER TABLE GlobalSession ALTER COLUMN CookieValue varchar(32)
GO
```



## 1.6 Adding/Modifying Indexes On The GlobalSessions And MVTestVariablePage Tables

To improve performance of your OMS 1.x implementation the indexes on the GlobalSession and MVTestVariablePage tables need to be added, or modified.

This section provides a script for handling this task.

### Note

Section 1.5 - Check GlobalSession.CookieValue DB Storage Length needs to be performed prior to running this task.

### 1.6.1 Required Skills

- A working knowledge of running T-SQL scripts.

### 1.6.2 Symptoms

- Performance issues related to running reports over large amounts of data.

### 1.6.3 How to Add/Modify the Indexes on the GlobalSessions and MVTestVariablePage tables

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click on the OMS database and select **New Query**.

```
Execute the following script:
USE [db name here]

--- Note, the GlobalSessionCookieValueLength.sql script
--- needs to run before this script to set the [dbo].[GlobalSession].
--- [CookieValue] length can be found in the OMS Tuning Guide,
--- section 1.5

-- Create a NONCLUSTERED INDEX IX_CookieValue on
-- the GlobalSession table

CREATE NONCLUSTERED INDEX [IX_CookieValue] ON [dbo].[GlobalSession]
( [CookieValue] ASC )
WITH (
    PAD_INDEX = ON,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    FILLFACTOR = 80
)
ON [PRIMARY]

--- Create a NONCLUSTERED INDEX IX_IP_IPAddressFrom on
--- the IP table (include IpId)

CREATE NONCLUSTERED INDEX [IX_IP_IPAddressFrom] ON [dbo].[IP]
( [IPAddressFrom] ASC )
INCLUDE ( [IpId] )
WITH (
    PAD_INDEX = ON,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
```

```

    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    FILLFACTOR = 90
)
ON [PRIMARY]

-- Delete and recreate the IX_VisitorIdentification INDEX
-- to increase FILL FACTOR, turn on PADDING, and use TEMPDB
-- space for sorting

IF EXISTS (
    SELECT * FROM sys.indexes
    WHERE object_id = OBJECT_ID (
        N'[dbo].[GlobalSession]') AND
        name = N'IX_VisitorIdentification_Globa')
DROP INDEX [IX_VisitorIdentification_Globa] ON
    [dbo].[GlobalSession]
    WITH ( ONLINE = OFF )

CREATE NONCLUSTERED INDEX [IX_VisitorIdentification_Globa] ON [dbo].[GlobalSession]
    ( [VisitorIdentification] ASC )
INCLUDE ( [GlobalSessionId] )
WITH (
    PAD_INDEX = ON,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = ON,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    FILLFACTOR = 80
)
ON [PRIMARY]

-- Create NONCLUSTERED INDEXES on
--- the [dbo].[MVTestVariablePage] table

CREATE NONCLUSTERED INDEX [IX_TestVariablePage_ID] ON [dbo].[MVTestVariablePage]
    ( [MVTestVariablePageId] ASC )
INCLUDE ( [PageId] )
WITH (
    PAD_INDEX = ON,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    FILLFACTOR = 80
)
ON [PRIMARY]

CREATE NONCLUSTERED INDEX [IX_TestVariablePage_PageID] ON [dbo].[MVTestVariablePage]
    ( [PageId] ASC )
INCLUDE ( [MVTestVariableId] )
WITH (
    PAD_INDEX = ON,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    FILLFACTOR = 80
)
ON [PRIMARY]

GO

```

## 1.6.4 Sitecore Recommendation

Sitecore recommends that the indexes be added to the GlobalSession and MVTestVariablePage tables to reduce the amount of time required to run reports over large amounts of data.

## Chapter 2

# Database Properties

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

This chapter contains the following section:

- Compatibility Level Set to SQL Server 2008 (100)
- Recovery Model Set to Simple
- Auto Close Property Set to False
- Auto Shrink Property Set to False
- Set Initial Size Value before Inserting Data
- Set Autogrowth Property before Inserting Data
- Server Requirements for OMS Database
- OMS Report Timeout Settings
- Connection String Parameters

## 2.1 Compatibility Level Set to SQL Server 2008 (100)

Compatibility Level effects SQL syntax and query parsing, and should have no impact of performance. Setting the Compatibility Level to a value of SQL Server 2008(100) will take advantage of new T-SQL features, which are used in many of the scripts / commands used in the OMS Performance Tuning Guide.

### 2.1.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.1.2 Symptoms

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

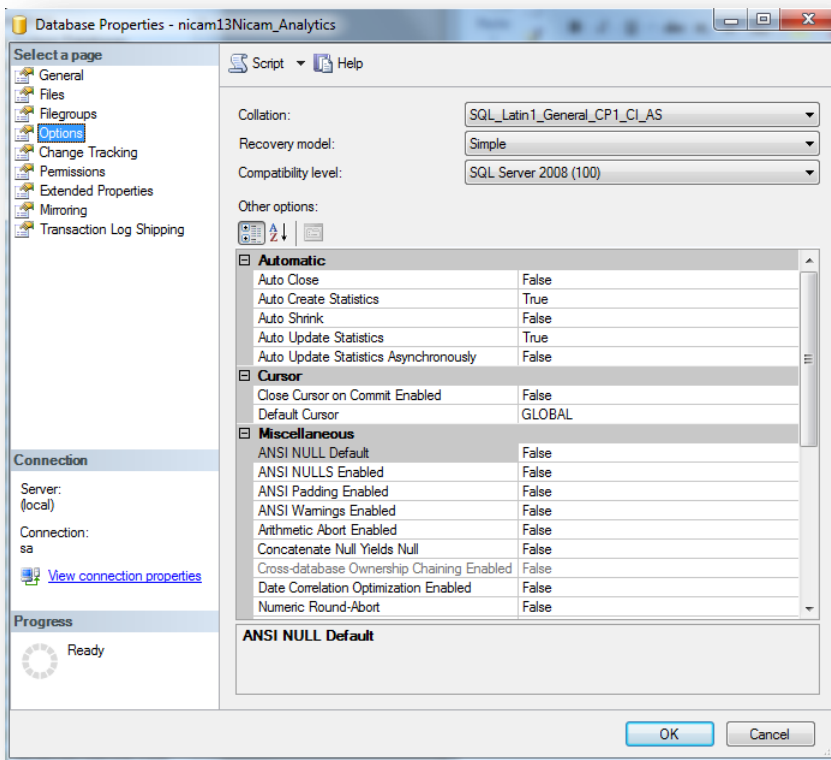
### 2.1.3 Checking the Database Compatibility Level

To check for the value of the *Database Compatibility Level*:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the OMS 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:



## 2.1.5 Sitecore Recommendation

Sitecore recommends that the *Compatibility Level* property be set to *SQL Server 2008(100)*.

## 2.1.6 How to Solve

To set the *Compatibility Level* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the OMS database, and then click **Properties**.
3. Select the *Options* page and make sure that the *Compatibility Level* is set to *SQL Server 2008(100)*.

## 2.2 Recovery Model Set to Simple

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 OMS database.

### 2.2.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.2.2 Symptoms

- Longer times required to recover the database.

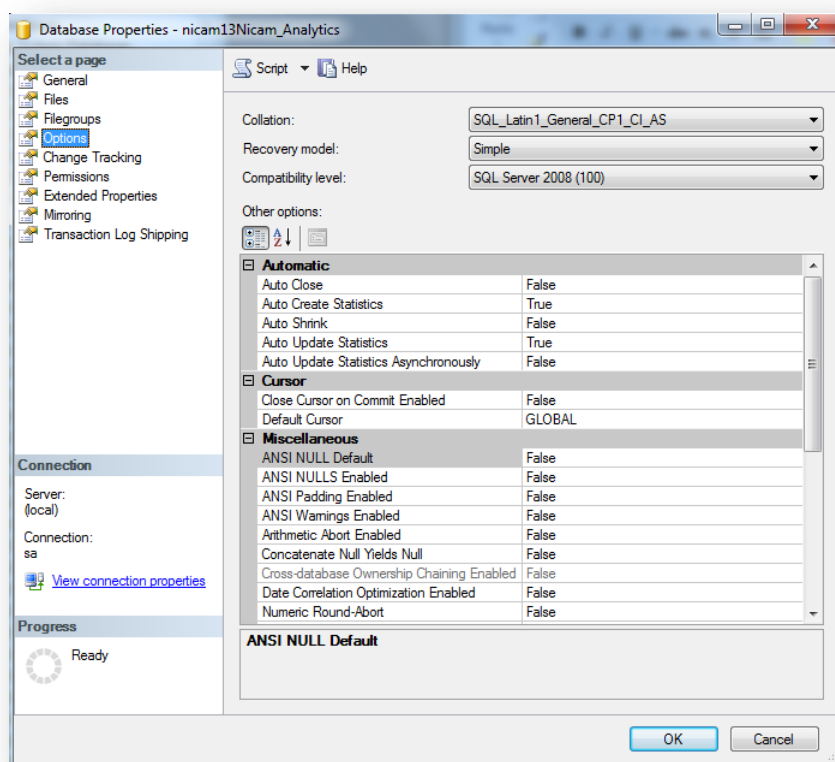
### 2.2.3 Checking the Recovery Model

To check for the value of the Recovery Model:

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

## 2.2.4 Understanding the Results

The output will look like this:



## 2.2.5 Sitecore Recommendation

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

## 2.2.6 How to Solve

In order to set the *Recovery Model* property:

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



## 2.3 Auto Close Property Set to False

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.

### 2.3.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.3.2 Symptoms

- Longer times required to connect to the database.

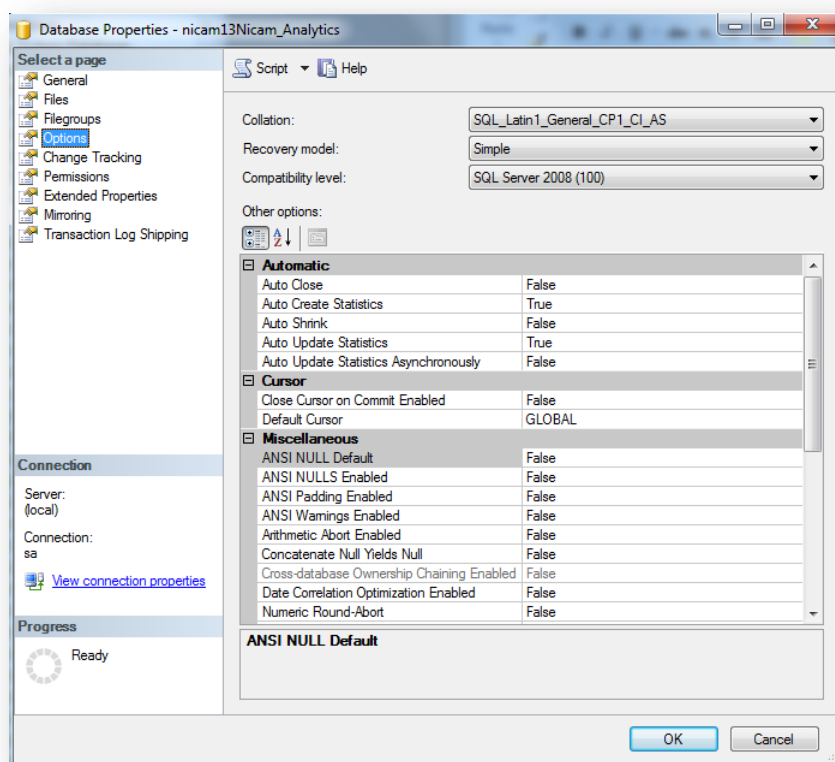
### 2.3.3 Checking the Value of the Auto Close Property

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

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

## 2.3.4 Understanding the Results

The output will look like this:



## 2.3.5 Sitecore Recommendation

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

## 2.3.6 How to Solve

In order to set the *Auto Close* property:

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

## 2.4 Auto Shrink Property Set to False

The *Auto Shrink* property has the downside:

- 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.

### 2.4.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.4.2 Symptoms

- Performance degradation.

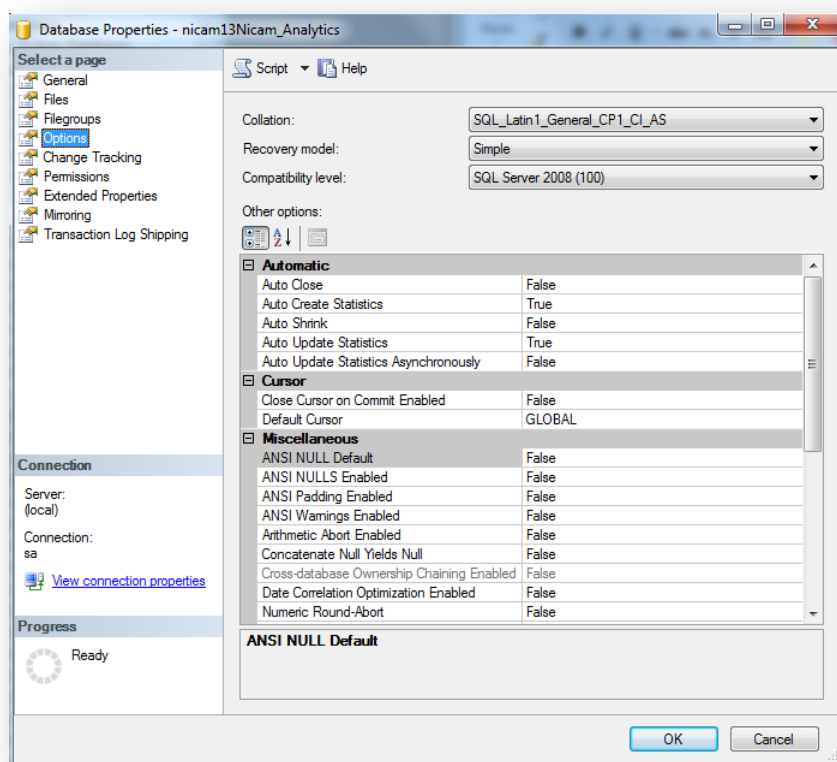
### 2.4.3 Checking the Value of the Auto Shrink Property

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

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

## 2.4.4 Understanding Results

The output will look like this:



## 2.4.5 Sitecore Recommendation

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

## 2.4.6 How to Solve

In order to set the *Auto Shrink* property:

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

## 2.5 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.

### 2.5.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.5.2 Symptoms

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

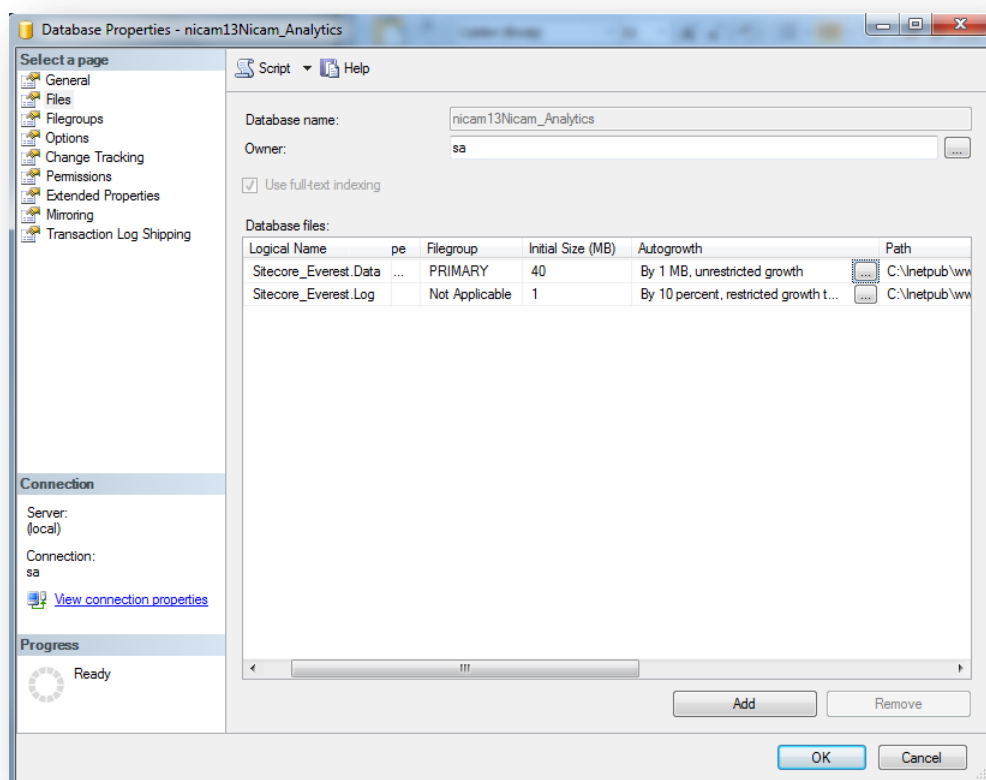
### 2.5.3 Checking the Initial Size Value

To check the *Initial Size* value:

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

## 2.5.4 Understanding the Results

The output will look like this:



## 2.5.5 Sitecore Recommendation

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

## 2.5.6 How to Solve

In order to set the *Initial Size* value:

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

## 2.6 Set Autogrowth Property before Inserting Data

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.

### 2.6.1 Required Skills

- A working knowledge of SQL Server 2008 Management Studio.

### 2.6.2 Symptoms

- Performance degradation.

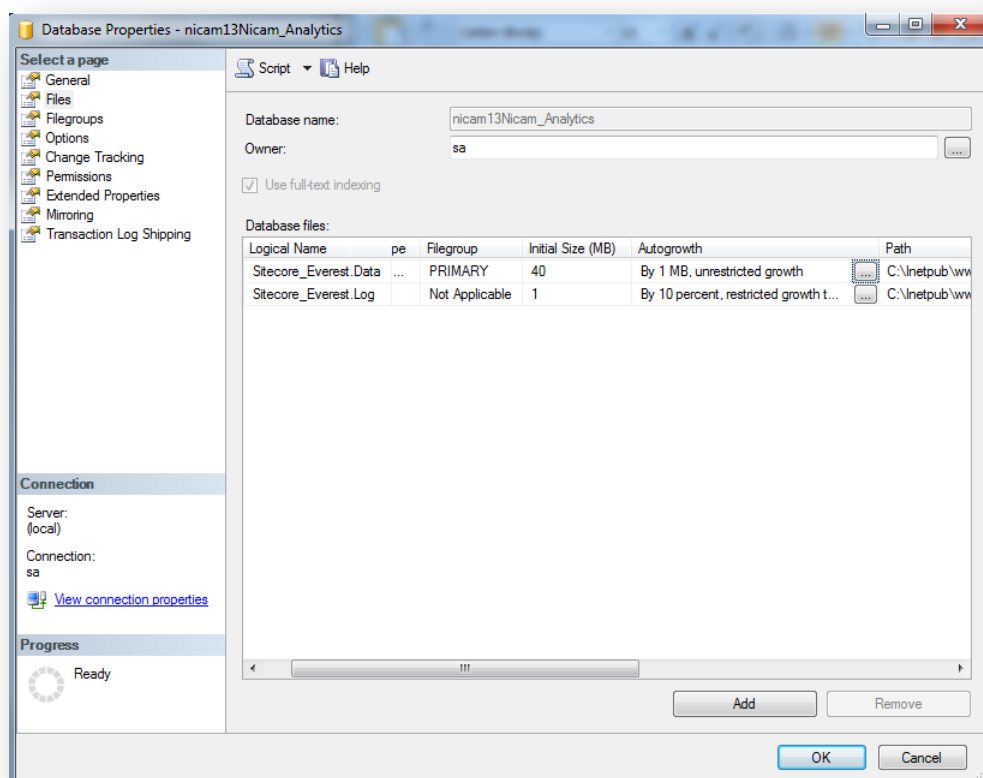
### 2.6.3 Checking the Value of the Autogrowth Property

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

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

## 2.6.4 Understanding the Results

The output will look like this:




## 2.6.5 Sitecore's Recommendation

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

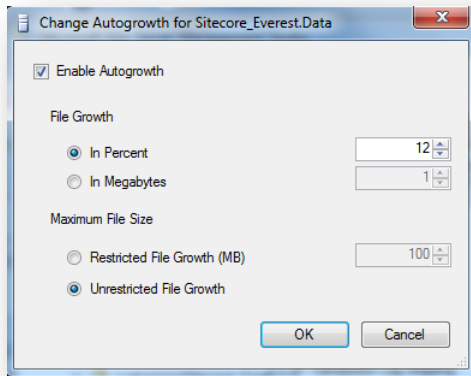
## 2.6.6 How to Solve

In order to enable and set the *Autogrowth* property:

1. Launch **SQL Server Management Studio**.
2. In the **Object Explorer**, right click the OMS 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:



## 2.6.7 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 OMS database should be set as large as possible. This will help reduce the frequency with which autogrowth occurs.

## 2.7 Server Requirements for OMS Database

This check is used to determine if the server configuration that you use to host the OMS 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 OMS Server Requirements on the SDN:

### 2.7.1 Required Skills

- A working knowledge of system infrastructures.

### 2.7.2 Symptoms

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

### 2.7.3 Sitecore Recommendation

Sitecore recommends the following minimum server requirements for the OMS 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 or 2008
  - 64 bit version recommended
- MS SQL Server 2008 or SQL Server 2005
  - SQL Server 2008 is strongly recommended due to improvements in Query Plans and memory usage
  - 64 bit version recommended

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	2005
	+1	-1

Score \_\_\_\_\_

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

Score \_\_\_\_\_

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

Score \_\_\_\_\_

## Hardware Requirements

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

Score \_\_\_\_\_

CPU	1 core	2 core	4 core	8 core	16 core +
<b>Standalone Server</b>	-1	0	+1	+2	+3
<b>Virtual Server</b>	-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\_\_\_\_\_

## 2.8 OMS Report Timeout Settings

The OMS report timeout settings should not be changed before you complete all the tasks that have been explained earlier in the OMS Tuning Guide.

Each OMS report has an associated \*.mrt file that specifies the parameters for running the report. One of those parameters is the `<CommandTimeout>XX</CommandTimeout>` — this specifies the amount of time — in seconds — that a report has to run before timing out.

The default `CommandTimeout` for all reports is 30 seconds.

### 2.8.1 Required Skills

- A working knowledge of Sitecore Analytics \*.mrt report files.

### 2.8.2 Symptoms

- After completing all the tasks in the OMS Tuning Guide, reports are timing out.

### 2.8.3 Checking the OMS Report Timeout Settings:

To check the OMS Report Timeout:

1. Navigate to the `<webroot>/sitecore/shell/Applications/Analytics/Reports` directory and open up the \*.mrt file for the report that is timing out.
2. Look at the `<CommandTimeout>XX</CommandTimeout>` setting.

### 2.8.4 Understanding the Results

The output will look like this:

```
<CommandTimeout>30</CommandTimeout>
```

The `CommandTimeout` parameter is the length of time (in seconds) that a report has to run before throwing a timeout error. The default setting is 30 seconds.

#### Note

In Sitecore 6.3 and earlier, a value of 0 will disable the `CommandTimeout`. In Sitecore 6.4, a value of 0 will take the value from the `Analytics.Reports.Timeout` setting.

### 2.8.5 Sitecore Recommendation

Sitecore recommends setting the `CommandTimeout` setting to a value of 120 to 300 (2 to 5 minutes), but only after all other prior OMS Tuning Guide tasks have been completed. If the reports are not experiencing timeouts, the `CommandTimeout` setting should be left at its default value of 30 seconds.

### 2.8.6 How to Solve

To set the `Connection String` parameters:

1. Navigate to the `<webroot>/sitecore/shell/Applications/Analytics/Reports` directory and open up the \*.mrt file for the report that is timing out.
2. Edit the `CommandTimeout` setting and save the file.
3. Clean up all the \*.dll files from the `\temp` folder of your Sitecore solution

## 2.9 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 OMS 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

### 2.9.1 Required Skills

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

### 2.9.2 Symptoms

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

### 2.9.3 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.

### 2.9.4 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=OMS_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.

## 2.9.5 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

## 2.9.6 How to Solve

In order 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.