Nimsoft Database Best Practices for MS SQL Server

Legal Notices

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Nimsoft LLC disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Nimsoft LLC shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Nimsoft LLC and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Nimsoft LLC as governed by United States and international copyright laws.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Nimsoft LLC's standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Trademarks

Nimsoft is a trademark of CA.

Adobe®, Acrobat®, Acrobat Reader®, and Acrobat Exchange® are registered trademarks of Adobe Systems Incorporated.

Intel® and Pentium® are U.S. registered trademarks of Intel Corporation.

Java(TM) is a U.S. trademark of Sun Microsystems, Inc.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Netscape(TM) is a U.S. trademark of Netscape Communications Corporation.

Oracle® is a U.S. registered trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of the Open Group.

ITIL® is a Registered Trade Mark of the Office of Government Commerce in the United Kingdom and other countries.

All other trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

Contact Nimsoft

Contact Technical Support

For your convenience, Nimsoft provides a central site where you can access all the information you need for your Nimsoft products.

At http://support.nimsoft.com/, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- Nimsoft Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about Nimsoft product documentation, you can send a message to support@nimsoft.com.

Document History

Version	What's new
1.0	February 21, 2012, version 1.0
1.1	March 29, 2012, added TNT2 data model appendix

Contents

Legal Notices 2
Contact Nimsoft 3
Document History 4

Chapter 1: Introduction 7

About this Guide 7
Background on the Nimsoft Information Store 7

Chapter 2: Nimsoft Information Store 9

NIS within Nimsoft Monitor 9
NIS Schema Overview 9
- Relationships between Tables 10
- Database Tables 10
- RN_QOS_DATA_ table columns 11
- RN_tables Indexes 11
- About Table Partitioning 11
- Manual Table Partitioning 12

Chapter 3: Prerequisites 14

Supported Database versions 14
Hardware Requirements 14
- About Database Performance 15

Chapter 4: General Best Practices 16

General 16
Storage 16
Backup 17
Periodic Maintenance 17

Chapter 5: Best Practices for SQL Server 18

Checklist 18
Network communication 18
Cluster environment 19
Windows configuration 19
Disks and SANs 20
SAN considerations 22
SQL server instance configuration 23
Database level configuration 24
Database maintenance and backup strategy 27
Additional Resources 28

Chapter 6: Performance Analysis 30

Checklist 30
Example Query (Wait Statistics) 30
Example Results....31

Chapter 7: Troubleshooting 32

Appendix A: Updating Table Indexes 33

Performing Analysis on QoS sample data tables 34

Appendix B: Advanced Indexing Topics 36

Advanced indexing when using report_engine/group_server 36
Option One 36
Option Two....36
Option Three 38
Best Practices for Clustered and Non-Clustered Indexing....39
Best Practices for Clustered Indexes 39
Best Practices for Non-clustered Indexes....39

Appendix C: SQL Tools and Scripts 40

Get db size info....40
Get_db_info_sqlserver 41
get_index_fragmentation....42
fix_index_fragmentation....43
find_missing_nodes_in_dynamic_views 43
sp_Generic_DefragindexesBasedOnFragmentation 46
shrink_transaction_log_sqlserver 52
Most costly unused indexes 53
Top Costly Missing Indexes 53
Indexes with the most contention 54
Most logically fragmented Indexes 54
Tables without clustered index 55
Tables with primary key, without clustered index 56
Objects with no indexes / Foreign keys that are not indexed....56
Number of indexes per table....58
Top SQL with highest CPU 58
Top SQL with highest I/O....59
Top SQL with highest Duration....59

Appendix D: TNT2 Data Model 60

Chapter 1: Introduction

About this Guide

This guide covers best practices for deployment, tuning, triage and maintenance of the Nimsoft database, also known as the Nimsoft Information Store (NIS), as deployed on MS SQL Server.

It draws together new and existing information from Nimsoft documentation, support articles, development tools, other internal sources as well as external sources—and organizes it into these major sections:

- General description of the NIS
- Prerequisites
- Best Practices
- Best Practices for SQL Server
- Performance analysis
- Troubleshooting
- Updating Table Indexes
- Advanced NIS indexing
- SQL Tools and Scripts

Microsoft's SQL Server platform was the first supported by Nimsoft. MySQL and Oracle were added later—similar guides for those databases will be released in the near future.

Note that this document does not attempt to fully document the programming interfaces, theory of operation, and structure of the NIS, rather it is intended to be useful for practical issues in database deployment, tuning, curing, and maintenance.

Background on the Nimsoft Information Store

The Nimsoft Unified Management solution requires a database to store the QoS, service level, configuration, alarm (optional) and other data that is collected, processed, and displayed by the system. The NIS is integral and critical to overall Nimsoft system operation and performance.

The NIS was originally introduced into the Nimsoft product to hold historical QoS data derived from raw data. This was to enable Service Level Monitoring (SLM) features, hence it became known as "NimsoftSLM" or just SLM.

With the introduction of expanded reporting and dashboard features in the Unified Management Portal (UMP), the SLM took on an expanded role, and was re-named the Nimsoft Information Store or NIS.

Originally, only Microsoft's SQL Server was supported, but in 2009 cross-platform support was introduced, adding support for MySQL and Oracle databases.

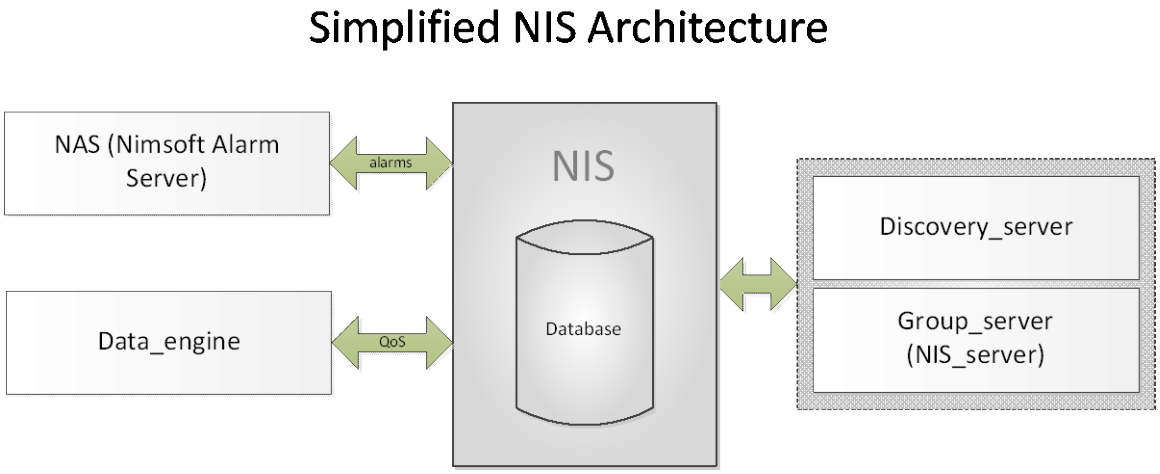
As with any OLTP application, the volume of stored data increases over time. Especially when scaled to the needs of large enterprise and managed service providers (MSPs), any database will require periodic maintenance and performance tuning. Nimsoft's customers, partners, and developers have generated a number of "best practice" procedures and tuning tips.

Two areas outside the scope of this document are database software upgrades and cross-platform data migration. We recommend you speak with your Nimsoft sales engineer regarding these activities.

Chapter 2: Nimsoft Information Store

These sections provide an overview of the schema and select components of the NIS.

NIS within Nimsoft Monitor



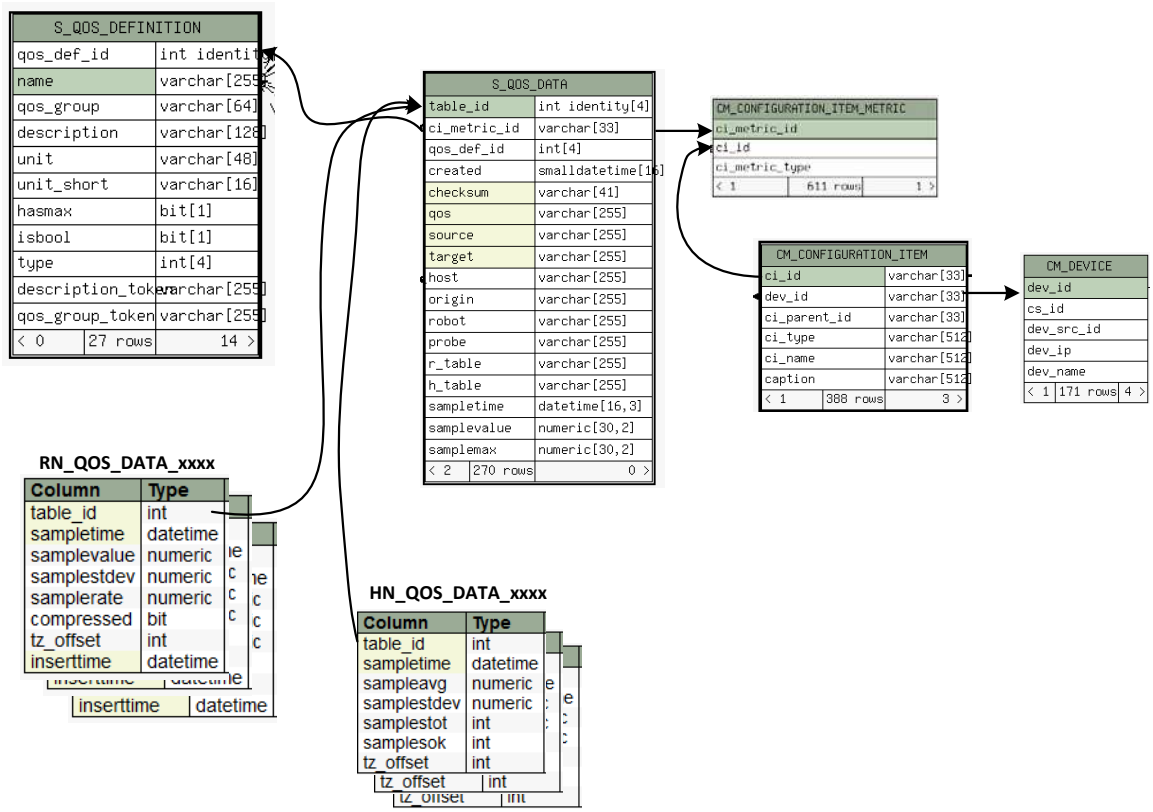
NIS Schema Overview

For a picture of how the NIS is structured, below is an abbreviated view from the SchemaSpy tool, listing the tables in the NIS database. The full view is many pages long, with hundreds of tables listed.

Table	Children	Parents	Columns	Rows
Account_			16	1
Address			18	0
AnnouncementsDelivery			7	30
AnnouncementsEntry			17	0
AnnouncementsFlag			5	0
AssetCategory			14	0
AssetCategoryProperty			9	0
AssetEntries_AssetCategories			2	0
AssetEntries_AssetTags			2	0
AssetEntry			25	127
AssetLink			9	
AssetTag			9	0
AssetTagProperty				14,854
AssetTag[illegible]_0004			8	890,855
[illegible]QOS_DATA_0005			9	890,855
RN_QOS_DATA_0006			9	892,579
RN_QOS_DATA_0007			8	890,856
RN_QOS_DATA_0008			8	890,856
RN_QOS_DATA_0009			9	890,856
RN_QOS_DATA_0010			9	890,856
RN_QOS_DATA_0011			9	890,857
RN_QOS_DATA_0012			9	890,857
RN_QOS_DATA_0013			9	94,946
RN_QOS_DATA_[illegible]			9	94,946
RN[illegible]ps			7	
[illegible]site			11	0
WikiNode			11	2
WikiPage			23	2
WikiPageResource			4	2
WorkflowDefinitionLink			10	0
WorkflowInstanceLink			10	0
WSRP_WSRPConsumer			9	0
WSRP_WSRPConsumerPortlet			7	0
WSRP_WSRPProducer			7	1
410 Tables			**3,343**	**18,729,661**

Relationships between Tables

(Implied relationships not shown)



Database Tables

These tables are of primary interest:

Table name or pattern	Type of data held
S_QOS_DATA	QoS data, used by data_engine
RN_QOS_DATA_xxxx	Raw QoS data—one RN_table for each qos_def_id
HN_QOS_DATA_xxxx	Aggregated (1hr interval) QoS data
CFG_*	ACE configuration information
CM_GROUP	NIS_Server/USM
CM_DEVICE	Discovery/configuration
CM_NETWORK	Discovery
CM_NIMBUS_*	Discovery/configuration
GRP_*	group_server, depricated
ump* and QUARTZ_*	UMP and LifeRay
*_	Liferay
*_t_*	Tmp, dashboard_engine
NAS*	Alarms*
tbnLogging	Sp logging
tbnVersion	Version info
S_SLA_* and S_SLO_*	Service Level Agreement info

***Note:** Alarm data is held in the separate Nimsoft NAS database. If the NIS-bridge feature of NAS is enabled, alarms are replicated from NAS to NIS. UMP accesses alarm data from the NIS using the NIS-bridge feature.

RN_QOS_DATA_ table columns

RN_QoS_Data_tables, as their name suggests, hold raw QoS data. QoS data is written once and never updated. It is aggregated by 1 hour periods and stored in HN_QoS_Data_tables_xxxx.

tableID	Sampletime	Samplevalue	Samplestdev	Samplerate	Samplemax	Compressed	Tz_offset	inserttime

Column Name	Description
tableID	unique identifier; key for looking up time series data
Sampletime	time the sample was taken
Samplevalue	QoS value
Samplestdev	standard deviation of the sample
Samplerate	Rate of sampling
Samplemax	Maximum sample value (e.g. 100%)
Compressed	Compressed (binary) (deprecated)
Tz_offset	time zone offset
Inserttime	Time for data insert to db (deprecated)

RN_tables Indexes

The default indexes on RN_tables (as of Nimsoft Server version 4.3x or data_engine v7.53) are optimized for writing data:

Index	Description
Idx0	clustered index (sampletime, table_ID)
Idx1	non-clustered index (table_ID, sampletime)
IdxSDP	used prior to Server v5.21, deprecated in v5.60

There is no primary key implemented on RN_QoS_DATA_tables, as both tableID and sampletime can be duplicated.

Prior to version 4.3x, RN_QOS_DATA_ tables used one clustered index on (tableID,sampletime). Updating pre-v4.3x table indexing is covered in the section "Updating Table Indexing."

With the use of report_engine or group_server, for example when generating on-demand Dynamic Views, a revised index approach is recommended. See the section "Advanced indexing for report_engine/group_server" for details.

About Table Partitioning

As RN_QoS_DATA_tables grow in size, the time needed to order and index them increases, slowing performance. Subdividing tables into multiple partitions offers several benefits:

- Partitioning allows data loads, index creation and rebuilding, and backup/recovery to occur at the partition level, rather than on the entire table.
- Partitioning improves query performance. In many cases, the results of a query can be achieved by accessing a subset of partitions, rather than the entire table.
- Partitioning can significantly reduce the impact of scheduled downtime for maintenance operations.

As of data_engine 7.85, NIS tables can be automatically partitioned if you are using Sql Server 2008 R2 Enterprise Edition. The partitioning scheme is a sliding window partition on sampletime, with one partition per day. If partitioning is enabled, data is aged out of the RN tables by dropping the old partitions rather than deleting.

Manual Table Partitioning

You may wish to apply partitioning to your large tables at first (use the query listed under Get db size info to find these). We define "large" as those having over 100m rows. Partitioning in a selective manner gives you more control over when tables are partitioned, as this process can take considerable to time to complete.

Use the following approach to apply manual partitioning.

1. Disable the data_engine probe
2. Disable the wasp probe
3. Disable the dashboard_engine probe
4. Run this for each table you wish to partition (one at a time is okay):

```
USE [NimsoftSLM]
GO

DECLARE     @return_value int,
            @pErrorMessage ndtLongString

EXEC  @return_value =
  [dbo].[spn_de_PartitionAdmin__PartitionTable]
            @pTableName = N'RN_QOS_DATA_0003', --
  run for each table
            @pCurrentTime = N'2012-02-21
  11:28:49.883',
            @pRawAge = 365,
            @pRawAgeExtra = 30,
            @pLogLevel = 5,
            @pErrorMessage = @pErrorMessage OUTPUT

SELECT      @pErrorMessage as N'@pErrorMessage'

SELECT      'Return Value' = @return_value

GO
```

After this has been done for the larger tables, check the box in the GUI which will enable partitioning on the rest of the tables.

Use this code to see what has been partitioned:

```
USE [NimsoftSLM]
GO

DECLARE     @return_value int,
            @pErrorMessage ndtLongString
```

```
EXEC  @return_value =
[dbo].[spn_de_PartitionAdmin__List]
            @pMode = list,
            @pErrorMessage = @pErrorMessage OUTPUT

SELECT      @pErrorMessage as N'@pErrorMessage'

SELECT      'Return Value' = @return_value

GO
```

Be sure to bring the system back into operation by re-enabling the following probes:

4. data_engine
5. wasp
6. dashboard_engine.

Chapter 3: Prerequisites

This guide assumes that Nimsoft Monitor and the NIS database are installed and running. We recommend you review the requirements and prerequisites for proper NM Server and database installation--see the section on "Nimsoft Server Pre-Installation" in the *Nimsoft Server Installation Guide*. This document is available from the downloads section at http://support.nimsoft.com.

Supported Database versions

The following database versions are supported on these operating systems:

Note: To take advantage of partitioning, you need to use Sql Server 2008 R2 Enterprise Edition

Database	Supported Operation System
Microsoft SQL Server 2005 , 2008, and 2008 R2	Windows Server 2003 and 2008

Hardware Requirements

Nimsoft always recommends deploying the database on a dedicated physical server.

When specifying hardware for the database server, generously configure the host machine and its associated storage facilities:

CPU	Multi-CPU (multi-core Xeon class or similar), 3.00 Ghz clock or better
Memory	(see chart below)*
OS HDD	50 GB minimum
Log file HDD	250 GB minimum
Data HDD	500 GB minimum, 1TB and RAID 10 recommended

*Based on estimated overall size of the database (which is a function of the number of QoS messages/min written, the configured data aggregation parameters and retention interval length):

Database size** after retention interval has been reached	Processor cores	RAM (GB)
Up to 50GB	2	4
50 to 150GB	4	8
150GB to 400GB	8	16
400GB to 600GB	8	28
600GB to 2TB	16	72
2TB or more	32	128

**Your Nimsoft Technical Account Manager has a spreadsheet that can provide estimates for these numbers, based on monitored devices, QoS metrics, and sampling rates.

About Database Performance

Relational database server performance is heavily affected by disk I/O performance and server bus bandwidth. Crowded VM hosts, clusters, or heavily shared storage or VM environments are not recommended for hosting the Nimsoft NIS database.

Nimsoft recommends starting with at least 1TB of RAID 10 storage for the NIS database. Also consider spreading the database files across multiple disks (LUNs) to improve I/O performance. Choose drive subsystems with low latency and seek times, high spindle speeds and high interconnect bandwidth.

Also, data redundancy/synchronization model needs to be considered on an on-going basis, taking into account the growth of the database. Selecting the right storage solution is beyond the scope of this document--we recommend you discuss this with your storage vendor/VAR/consultant.

Chapter 4: General Best Practices

This section covers operational NIS best practices There are categories for:

- General
- Storage
- Backup
- Periodic maintenance

General

Best Practice	Comments
Read and observe documented pre-requisites and pre-install information	Available in the section "Server Pre-Installation" in *NM Server Installation Guide*
Always make a backup of your database before upgrading major Nimsoft components (e.g. Server and UMP)	Some upgrades contain a non-reversible upgrade script that changes the database structure of some tables
Run get_db_info_sqlserver on a regular basis	Establish a baseline so that system changes can be easily seen and Nimsoft support can quickly respond to issues
Use stored procedure ' sp_who2 active'	Provides information on status
Set up periodic re-indexing	Use Nimsoft data_engine
Set up regular defragmentation	Use MS SQL Server agent
Check database size	Use sp_spaceused
Check index fragmentation on a regular basis and fix (SQ Server)	get_index_fragmentation fix_index_fragmentation . Also: DefragIndexesBasedon Fragmentation
Check index fragmentation (MySQL and Oracle)	sp_Generic_DefragIndexesBasedOnFragmentation
Reduce transaction log size on a regular basis	shrink_transaction_log_sqlserver
Check for missing nodes in Dynamic Views	Find_missing_nodes_in_dynamic_views

Storage

Best Practice	Comments
Determine an overall storage strategy	Discussion with storage vendor

Backup

Best Practice	Comments
Plan and schedule regular backups of the database	
Test Restore operation before it is needed	
Ensure sufficient disk capacity for backups	
Document Backup and Restore procedures	

Periodic Maintenance

Best Practice	Comments
Identify and remove index fragmentation	Use get_index_fragmentation & fix_index_fragmentation tools. Also DefragIndexesBasedon Fragmentation
Identify skewed and outdated index and column statistics and make sure they are representative and current	Index statistics are used by the SQL Server query optimizer to help it determine if, and when, an index should be used when executing a query.
Identify and create missing indexes	Use tools such as the SQL Server Database Engine Tuning Wizard to help select missing indexes based on Profiler traces you have collected. Or hand tune indexes based on your analysis of the execution plans of critical queries. A third option is to use a combination of the first two.
Identify and remove unused indexes	Unused indexes add unnecessary overhead because they have to be modified every time data is modified in a table.
Create and monitor index maintenance jobs	Create scripts necessary to perform automated index maintenance, and then schedule them with the SQL Server agent. Also, configure SQL Server to notify you if any of these jobs fail.
Database and log file protection and management	
tempdb maintenance	
Data corruption detection	
Performance monitoring	

Chapter 5: Best Practices for SQL Server

This section provides a checklist of values and properties to check when setting up and deploying Microsoft SQL Server.

Checklist

This checklist is hierarchical in method, starting with hardware and OS settings, then SQL Server instance, then the NIS database and its maintenance.

- Network communication
- Cluster Environment
- Windows configuration
- Disks and SAN
- SQL Server instance
- Database level configuration
- Database maintenance and backup strategy
- Additional Resources

Network communication

Applies to (Windows versions)	Item	Recommendation
All	NIC full duplex	Network adapters and switch ports should have matching duplex levels or transfer speed settings. Full duplex provides better performance. “Maximize Data Throughput for Network Applications” should be set. http://support.microsoft.com/?scid=kb;EN-US;Q325487#top
All	Network settings	Latest basic input/output system (BIOS) update for the server should be installed. Latest firmware update for the network adapter should be installed. Latest driver update for the network adapter MUST be installed http://support.microsoft.com/kb/942861#top
All	NetBIOS and Server Message Block enabled?	Disable NetBIOS and Server Message Block http://msdn.microsoft.com/en-us/library/ms144228(SQL.90).aspx#disabled_protocols Important:Make sure NetBIOS is not in use.

Cluster environment

Applies to (Windows versions)	Item	Recommendation
All	Cluster nodes hardware	Cluster nodes should have nearly identical hardware on all cluster nodes to simplify configuration and eliminate potential compatibility problems.
All	Memory adjustment	In an Active/Active/… environment, max memory for the SQL Server instances should be set in a way that the total memory in the weakest node is split between the nodes. This will ensure that when all instances failover to one node, they will be able to that quickly and with no memory issues.

Windows configuration

For more information refer to: Performance Tuning Guidelines for Windows Server 2008 R2 or Performance Tuning Guidelines for Windows Server 2003 .

Applies to (Windows versions)	**Item**	**Recommendation**
All	Latest service pack	Implement the latest service pack and hotfixes
All	64 bit hardware and software	Recommended for higher performance
32-bit OS (Win 2000 ADV , win2003 ALL) , SQL server 2000 ENT Systems with 4 GB RAM only	Address more than 2 GB RAM	Add /3GB switch to boot.ini file to force the OS reserve to be only 1GB, while allowing applications (SQL Server) to use 3GB. http://support.microsoft.com/kb/274750
32-bit OS Enterprise edition only (Not WIN 2000) Systems with > 16 GB RAM only	Address more than 16 GB RAM	Add the /PAE switch in boot.ini to allows OS to access physical memory beyond 4GB. This switch is supported in Windows Server 2003, 2008 Enterprise and Datacenter editions. http://support.microsoft.com/kb/274750
32-bit OS Enterprise edition only (Not WIN 2000) Systems with between 4 & 16 GB RAM only	Address more than 4 GB RAM	Add *both* /PAE and /3GB switches in boot.ini to allow OS to access physical memory beyond 4GB. These switches are supported in Windows Server 2003, 2008 Enterprise and Datacenter editions. http://support.microsoft.com/kb/274750
64-bit OS , SQL server 2005 ENT Edition, SQL Server	Enable lock pages in memory	Reduce paging of buffer pool memory in the 64-bit version of SQL Server 2005. http://support.microsoft.com/kb/918483

Applies to (Windows versions)	Item	Recommendation
Standard Edition SP3 CU4 and higher		
All	Paging file	Pagefile should be 1~1.5 times the amount of RAM and should NOT be placed on a drive that contains database files. Important: It is recommended to create multiple page files on different disk partitions beside C:\ or even different disk subsystems for performance reasons.
All	System properties > Advanced setting	Processor scheduling: Select "Background services". Memory usage: Select "Programs".
Windows 2003,2008 with SQL 2005 or higher	Database Instant File Initialization enabled?	A way to prevent data file (not log file) Create and Grow operations zero- initializing new space before allowing it to be used. This dramatically speeds up these operations as zero- initialization can take a *long* time for large files. To enable the feature : From Local Policies >User assignments, “Perform volume maintenance tasks” entry and then add the SQL ServerService account or the local group SQLServerMSSQLUser$instancename.
All	Unnecessary Services / applications	Number of running apps and services should be minimal. Unnecessary services should be stopped and disabled (Messenger, wireless configuration ...etc)
All	Anti-virus?	Best Practice for SQL Server is *not* installing anti-virus on a dedicated SQL Server environment. If necessary then exclude MDF, NDF and LDF file extensions from being scanned. http://support.microsoft.com/kb/309422.

Disks and SANs

This section discusses the general configuration of the disk system (for detailed information regarding the placement of database files, refer to Database level configuration)

Applies to (Windows versions)	Item	Recommendation
All	NTFS Allocation Unit Size	Use : fsutil fsinfo ntfsinfo <DriveLetter>: SQL Server writes in 8k pages and the read-ahead buffer is 64k so format any disks which may be used for SQL Server data (data, logs, and tempdb) using a 64k block size. Summary: NTFS block size - 64K Stripe size - 64 KB or 256 KB Disk offset - 64K Useful links:

Applies to (Windows versions)	Item	Recommendation
		http://blogs.msdn.com/psssql/archive/2008/02/06/how-it-works-how-does-sql-server-backup-and-restore-select-transfer-sizes.aspx http://sqlblogcasts.com/blogs/ssqanet/archive/2008/04/28/sql-server-2005-and-disk-drive-allocation-unit-size-to-64k-any-benefit-or-performance.aspx http://blogs.msdn.com/johnhicks/archive/2008/03/03/sql-server-checklist.aspx http://technet.microsoft.com/en-ca/library/cc966414.aspx#EAOAC http://support.microsoft.com/default.aspx?scid=kb;EN-US;929491 Disk alignment instructions: http://technet.microsoft.com/en-us/library/aa995867.aspx **Always consult with your storage vendor regarding any changes**
2000,2003 (not 2008)	Volume alignment	NTFS volumes should be aligned. This can be done by using diskpar.exe or DiskPart.exe. DiskPart.exe is a disk configuration utility that is built into Windows; in Windows 2003 version SP1 or greater, DiskPart.exe contains an ALIGN option that can be used to align volumes. http://support.microsoft.com/default.aspx/kb/929491 http://blogs.msdn.com/jimmymay/archive/2008/12/04/disk-partition-alignment-sector-alignment-for-sql-server-part-4-essentials-cheat-sheet.aspx **Always consult with the storage vendor before making any changes**
All	Disks and LUNs	Storage can be divided into LUNs and the server will access one or more of these units as a partition or drive. Consider dedicating entire disks in the SAN to separate LUNs thereby isolating the I/O on each drive to the particular activity the drive will encounter. This is more important for log files where the I/O is sequential in nature--and any other disk activity (e.g. random read) can increase the log write latency. http://blogs.msdn.com/sqlcat/archive/2005/11/21/495440.aspx
All	Raid levels for SQL Server	There are 3 configurations that can be used for SQL server deployments (Do not consider using RAID 0) **RAID 1**: disk mirroring, provides a redundant, identical copy of a selected disk. All data written to the primary disk is written to the mirror disk. RAID 1 provides fault tolerance and generally improves read performance but may degrade write performance. **RAID 5:** striping with parity, this level stripes the data in large blocks across the disks in an array and it writes the parity across all the disks. Data redundancy is provided by the parity information. Striping with parity offers better performance than disk mirroring (RAID 1). However, when a stripe member is missing, read performance is decreased, for example, when a disk fails. **RAID 10:** mirroring with striping. RAID 10 uses a striped array of disks that are then mirrored to another identical set of striped disks. RAID 10 provides the performance benefits of disk striping with the disk redundancy of mirroring. RAID 10 provides the highest read-and-write performance of any

Applies to (Windows versions)	Item	Recommendation
		one of the other RAID levels, but at the expense of using two times as many disks. **General – for SQL Server:** Raid 10 should be used for everything, if possible, and not too expensive Raid 5 will be the second best option for database data files Raid 1 will be the next best option for T-Log files.

SAN considerations

Important: Discuss the appropriate values for these settings with your storage vendor.

HBA drivers: ensure that you are using the recommended drivers for your particular storage array. This should be located on the SAN vendor's Web sites for download.

HBA Queue depth settings: SQL Server applications are generally I/O-intensive, with many concurrent outstanding I/O requests. The default depth value is of 8 to 32 for the major HBA vendors. According to a Microsoft test (http://technet.microsoft.com/en-us/library/cc966412.aspx#EEAA) they have seen substantial gains in I/O performance when increasing this to 64 or even higher.

When Queue Depth is set too low, a common symptom is increasing latency and less-than-expected throughput given the bandwidth between host/storage and the number of spindles in a particular configuration

More information can be found here:
http://sqlblog.com/blogs/linchi_shea/archive/2007/09/18/sql-server-and-sans-the-queuedepth-setting-of-a-host-bus-adapter-hba.aspx

SQL server instance configuration

Applies to (SQL server versions)	Item	Recommendation
All	Latest service pack	Latest service pack and hotfixes. http://support.microsoft.com/kb/916287
All	SQL installed on Domain controller?	SQL server should never be installed on a domain controller
All	Dedicated machine for SQL Server?	Best Practices imply that SQL Server should be installed on its own dedicated host machine
All	Named instances are using dynamic ports?	Assign static ports to named instances of SQL Server so that SQL browser doesn't have to look up the current dynamic port. More info about how to check the value at: http://blogs.msdn.com/sqlserverfaq/archive/2008/06/02/how-to-change-the-dynamic-port-of-the-sql-server-named-instance-to-an-static-port-in-a-sql-server-2005-cluster.aspx
All	Server allowed protocols (TCP/IP, VIA, NPs, etc.)	Limit the supported protocols.
All	Max Server memory value	SQL server shouldn't be allowed to address *all* server memory, about 1.5-2 GB should be left for the OS. 2,000 MB is already the MAX for SQL 2000 STD http://technet.microsoft.com/en-us/library/ms181453.aspx
All	Affinity mask	Unless there are multiple instances and the CPUs are distributed on them, leave this as default. http://technet.microsoft.com/en-us/library/ms189435.aspx
All	Max Degree Of Parallelism (Max DOP)	The wait statistics IN SQL Server should be examined (http://msdn.microsoft.com/en-us/library/ms190732.aspx). If there is contention due to parallelism, parallelism should be disabled (either per query: Max DOP = 1 or globally for SQL Server sp_configure) : http://msdn.microsoft.com/en-us/library/ms181007.aspx
All	Boost SQL server priority	Be careful here as this can degrade the performance of other applications running on the same Server with SQL Server. Set it ON only if the box is dedicated to SQL Server only. **Important: Don't enable this for clustered servers.**
All	Login auditing	Usually "**failed logins**" will be audited. Not recommended to audit both successful and failed logins unless necessary, as this will increase the size of the SQL Server error log. Note: Changing the audit level requires restarting the service.
2005,2008	C2 Audit	Should only be enabled for an essential reason because of the impact it makes on system performance and the disk space it requires.

Applies to (SQL server versions)	Item	Recommendation
All	Recycle SQL Server Errorlogs?	SQL Server and SQL Server Agent error logs may grow very large. It is a good practice to recycle them on a regular basis: http://msdn.microsoft.com/en-us/library/ms182512.aspx http://technet.microsoft.com/en-us/library/ms178310.aspx
All	Database files location setting	Database default *data* location should be changed to point to a dedicated disk for data files and *logs* to a location dedicated for log files (optimized for write operations).
2008 ENT Edition	Compress backup	Should be enabled if there are large backups and necessary to reduce space occupied by the backup files.
2005,2008 on systems with more than 16-32 GB RAM	Enable to large-page allocations for the buffer pool	Enable **Trace flag 834** for high performance workloads on servers with high-end resources (CPU and memory specifically) to use Microsoft Windows large-page allocations for the buffer pool.

Database level configuration

Applies to (SQL server versions)	Item	Recommendation
All	Database files placement	• Log files should always be kept on RAID 1+0. Second best is Raid 1. • If possible also keep data and index files on RAID 1+0, but if this is too expensive (needs at least 4 disks and utilizes half the space), use Raid 5 as the second choice, especially if Read operations are expected more than write operations (which is usually the default with OLTP databases). • Because of the heavy use of the TempDB, RAID1+0 is the preferred solution for this database. Raid 5 is the second choice. **At a minimum always separate data files from log files on separate physical disks.** **If possible, separate tempdb files off to a dedicated disk.**
All	User/system DBs on system drive?	Placing database files on the same drive (logical) as the system drive affects database performance since the system will be busy serving system files from the same drive. Also, if the database grows significantly (tempdb for example), it may fill the system drive—with a severe effect on the system (this can even bring the server down). All system/user databases should reside on drives dedicated for SQL server.
All	Number of Tempdb datafiles	SQL server 2000 : 1 datafile / processor core SQL 2005, 2008 : 0.5 datafile / processor core
All	Database files filegrowth	• Filegrowth shouldn't be very large because user activity will have to wait for the file operation to complete.

Applies to (SQL server versions)	Item	Recommendation
		• Filegrowth shouldn't be very small to avoid issue from the file being filled up. • Growth should never be in terms of % but in terms of MBs.
All	Filegroups (FG) are used for large databases?	FGs should always be used for large databases (>50 GBs) for better recovery and organization of database objects. A best practice implementation is as following: • Primary: Contains only the MDF--no additional data goes into the .MDF file. It stays very small and contains only system objects; this will help with fast and partial recovery. • Secondary: create one (or more) data files on multiple disks, and then create (or move) tables to that filegroup. Also make this filegroup the default one so that any newly created objects go here. • Index (optional): create one (or more) data files on multiple disks, and then create (or move) indexes to that filegroup. This approach works if there is a large index and you want to move it to a separate disk. A separate filegroup can also be used for rarely accessed data and put on a separate, slower I/O path, in order to preserve fast RAID disks for highly used filegroups. There are DR benefits of having filegroups; refer to the blog here http://mssqltips.com/tip.asp?tip=1621
All	Primary FG is the default FG?	Other FGs that hold data should be default FGs so that new objects are created on secondary FGs
All	Database options are set?	Database settings should adhere to best practices. Some options are commonly altered (sometimes to wrong values), while others stay as default: • **Recovery model**: for DR purposes, FULL should be used to allow for point-in-time recovery. SIMPLE recovery may increase performance by reducing the cost of T-Log management by SQL Server. • **Auto Close:** Should NEVER be set to TRUE. • **Auto Shrink**: Should NEVER be set to TRUE. • **Auto Update Statistics**: Should ALWAYS be set to TRUE (except for very few exceptions, such as databases with extensive data loads. • **Page Verify**: Should ALWAYS be set to CHECKSUM (2005, 2008) and torn_page_detection must be selected in SQL server 2000. • **Parameterization** (2005,2008): should be set to "Forced" when there are applications that pass a large number of non-parameterized batches to lower procedure cache memory utilization and, therefore, have better overall throughput. More information: http://technet.microsoft.com/en-us/library/ms188124.aspx

Applies to (SQL server versions)	Item	Recommendation
All	Number of Virtual Log Files (VLFs) ?	Number of VLFs should not be more than 50, this impacts the performance of writing to the log file. http://sqlblog.com/blogs/linchi_shea/archive/2009/02/09/performance-impact-a-large-number-of-virtual-log-files-part-i.aspx Log files initial size should be large enough to reduce filegrowth occurrences and filegrowth value should be large enough to prevent excessive growth. See **script for getting VLFs**. **To reduce the VLF number:** Check the current size of T-Log **Databases in FULL recovery model:** 1 - Full backup (optional) 2 - TLOG backup (with truncate) 3 - Shrink the log file to… 10 megabytes? 4 - Grow the log file in 1 operation to its original size **Databases in SIMPLE recovery model:** 1 - Full backup (optional) 2 - Shrink the log file to… 10 megabytes? 3 - Grow the log file in 1 operation to its original size
2005,2008	Encryption used?	For critical data use encryption. There are performance implications to the usage of encryption functions. Make sure the encrypted columns have no indexes and are not included in WHERE clauses. SELECT * FROM [db].sys.symmetric_keys SELECT * FROM [db].sys.asymmetric_keys
2008	Data compression	If there is much concern about the space used by databases, compression can be implemented for less-accessed tables (especially large ones). Saved space should be measured first , sp_estimate_data_compression_savings Refer to the article here : http://www.mssqltips.com/tip.asp?tip=1582
2005,2008	Allow page locks for indexes	All indexes should have allowed page locks for better escalation management and the ability to be defraged or rebuilt. Get list of indexes without “allow page locks” Select * from db.sys.indexes where allow_page_locks=0

SQL query to get number of VLFs in databases (SQL Server 2005, 2008)

```
SET NOCOUNT ON
DECLARE @db nvarchar(250)

Create TABLE #loginfo
(
        FileId int , FileSize decimal( 28 , 0 ) ,
        StartOffset decimal( 28 , 0 ) , FSeqNo decimal( 28 , 0 ) ,
        Status tinyint , Parity tinyint , CreateLSN varchar( 40 ))

Declare  @VLFs TABLE (Db_name sysname , VLFS_count int)

DECLARE VLFS Cursor FOR
select  name  from  master.sys.databases  where  state_desc=  'online'
--2000
--select name from master..sysdatabases

OPEN VLFS

FETCH NEXT FROM VLFS INTO @db
WHILE @@FETCH_STATUS = 0
    BEGIN
        Insert into # loginfo
        Exec ('DBCC LOGINFO ('+@db+')')
        Insert into @VLFs
        select @db,count(*) from #loginfo
        Delete from @loginfo
        FETCH NEXT FROM VLFs INTO @db
    END
CLOSE VLFs
DEALLOCATE VLFs

select * from @VLFs order by Vlfs_count desc
```

Database maintenance and backup strategy

Applies to (SQL server versions)	Item	Recommendation
ALL	Recovery models	If the database is in the SIMPLE recovery model, the T-Log cannot be backed up and therefore there is no point-in-time recovery. In case of a failure, such as database corruption, the database will need to be restored from the latest FULL database backup/differential backup. If the database is on FULL recovery model, make sure that T-Log is being backed up, otherwise that T-log may grow indefinitely. Choose the database recovery model and backup strategy based on the database SLA. Make sure that database backups are copied off the server for DR. For better performance, don't backup the databases or T-logs to the same physical disk as the database or T-log files.

Applies to (SQL server versions)	Item	Recommendation
ALL	Index maintenance tasks	The Index Rebuild and Index Reorg default maintenance plans rebuild/reorganize all indexes blindly, no matter if they are fragmented or not. For better performance, it is strongly advised to reorganize/rebuild indexes based on their size and fragmentation level. This procedure can run even every day so that the reorg will be faster and lighter ("delta" defrag). Note that index rebuild is a resource consumer and may also increase the size of the T-Log (and therefore also of the T-log backups) SQL 2005 and higher: use ONLINE index rebuild as much as possible in Enterprise Edition when tables need to be available at all times.
ALL	Shrink database maintenance task	Never shrink databases or database files on a regular basis. If the file has grown once, it will most probably grow again. When a database file grows while transactions are running, this may degrade performance.
ALL	Update Statistics maintenance task	If Auto UPDATE Statistics option is turned on in the database and there is no massive insertion/update of data, this task is not required (which is true in most cases).
ALL	Maintenance and backup files cleanup	Don't forget to clean up old backup files according to the database SLA and the amount of free space on the backup drives. It is also necessary to purge the job and maintenance plans history from msdb. Large history tables in tempdb may cause contention and performance issues (even can lead to jobs failing).
ALL	Database Integrity checks	It is important to run integrity checks on the database. The earlier you find consistency issues, the better. Data corruption is usually very rare. Since this task is a resource consumer, it shouldn't be executed more than once a week.

Additional Resources

SQL Server Management Studio F1 Help
http://technet.microsoft.com/en-us/library/ms180294.aspx

Storage Top 10 Best Practices
http://sqlcat.com/top10lists/archive/2007/11/21/storage-top-10-best-practices.aspx

SQL Server 2005 Configuration Blog #2.doc
http://blogs.msdn.com/sqlcat/archive/2005/11/21/495440.aspx

Deploying SQL Server 2005 with SAN #3
http://blogs.msdn.com/sqlcat/archive/2005/11/17/493944.aspx

Storage Consideration for SQL Server 2005 DW environment
http://blogs.technet.com/vipulshah/archive/2008/04/21/storage-consideration-for-sql-server-2005-dw- environment.aspx

Physical Database Storage Design
http://technet.microsoft.com/en-us/library/cc966414.aspx

High Performance Data Warehouse with SQL Server 2005
http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/RelDWPerf.doc

Best Practices for Optimizing SQL Server in SAN Environments
http://www.symantec.com/community/article/4406/best-practices-optimizing-sql-server-san-environments-using-altiris- products

SQL server Clustering best practices
http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1197424_mem1,00.html

Common QA for deploying SQL Server in a SAN Environment
http://hosteddocs.ittoolbox.com/Microsoft-Siebel091704b.pdf

Chapter 6: Performance Analysis

This section provides a checklist for analyzing overall database performance, and can be used for continuous performance analysis and database tuning.

Checklist

Most issues listed in the right column link to a listing of the SQL query that will generate the desired output (valid for SQL Server only):

Performance analysis area	Issue to identify
Analyze Wait Statistics	Wait times (see example below)
Perform Index Analysis, identifying top issues->	Top costly unused indexes Top costly missing indexes Indexes with the most contention Most logically fragmented indexes Tables without a clustered index Tables with primary key, without clustered index Objects with no indexes Foreign keys that are not indexed Number of indexes per table
Identify top SQL queries according to high resource utilization->	Top SQL with highest CPU utilization Top SQL with highest I/O Top SQL with highest execute duration Heaviest stored procedures (SPs) for tuning (query TBD)

Example Query (Wait Statistics)

This query provides wait statistics that give a good indication of resource bottlenecks from an SQL Server perspective:

```
SELECT TOP 10
 [Wait type] = wait_type,
 [Wait time (s)] = wait_time_ms / 1000,
 [% waiting] = CONVERT(DECIMAL(12,2), wait_time_ms * 100.0
               / SUM(wait_time_ms) OVER())
FROM sys.dm_os_wait_stats
WHERE   wait_type   NOT   LIKE   '%SLEEP%'   and   wait_type   not   in
('SQLTRACE_BUFFER_FLUSH','DISPATCHER_QUEUE_SEMAPHORE','REQUEST_FOR_DE
ADLOCK_SEARCH',
'XE_TIMER_EVENT','FT_IFTS_SCHEDULER_IDLE_WAIT','XE_DISPATCHER_WAIT','
LOGMGR_QUEUE','CHECKPOINT_QUEUE','BROKER_TO_FLUSH')
ORDER BY wait_time_ms DESC;Index Analysis
```

Example Results

Abstracted from a production database (with parallelism enabled):

Wait type	Wait Time (ms)	% Wait Time
CXPACKET	781810	41.01
PAGEIOLATCH_EX	318017	16.68
PAGEIOLATCH_SH	307897	16.15
ASYNC_NETWORK_IO	112675	5.91
BACKUPIO	80461	4.22
ASYNC_IO_COMPLETION	51813	2.72
BACKUPBUFFER	49769	2.61
OLEDB	36811	1.93
WRITELOG	32093	1.68
BACKUPTHREAD	29272	1.54

After disabling parallelism (run DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR);) the resulting statistics show I/O contention:

Wait type	Wait Time (ms)	% Wait Time
PAGEIOLATCH_EX	1479	93.16
WRITELOG	81	5.15
OLEDB	10	0.65
ASYNC_NETWORK_IO	10	0.63
PAGEIOLATCH_SH	2	0.15
LCK_M_X	2	0.14
PAGEIOLATCH_UP	0	0.06
SOS_SCHEDULER_YIELD	0	0.03
PAGELATCH_UP	0	0.00
SQLTRACE_FILE_WRITE_IO_COMPLETION	0	0.00

Another view showing network and I/O contention:

Wait type	Wait Time (ms)	% Wait Time
ASYNC_NETWORK_IO	17994	20.33
BACKUPIO	14053	15.88
PAGEIOLATCH_EX	10654	12.04
ASYNC_IO_COMPLETION	10549	11.92
BACKUPBUFFER	10497	11.86
PAGEIOLATCH_SH	5940	6.71
PAGELATCH_EX	5633	6.36
BACKUPTHREAD	3511	3.97
OLEDB	3500	3.95
WRITELOG	2070	2.34

Chapter 7: Troubleshooting

This section provides best practices to triage and troubleshoot a distressed database.

Best Practice	Comments
Recognize a distressed database, learn the signs	One symptom is the data_engine queue backing up
Investigate what other processes are running	Sp_who2 active Activity Monitor (need more info on this)
Investigate index fragmentation	Use get_index_fragmentation & fix_index_fragmentation tools. Also DefragIndexesBasedon Fragmentation
Check disk subsystem(s) and drive failure	Performance impact to RAID Check RAID manager console
Resource constrained?	Task Manger (Windows) Top command in Linux and Solaris
Other jobs running	Task Manager (Windows) Top command in Linux and Solaris

Appendix A: Updating Table Indexes

The data_engine version 7.53 (shipped with NM Server 4.3x) changed the set of indexes created on newly-created QoS sample data tables RN_QOS_DATA_x. By default, existing RN_QOS_DATA-tables were left with the old index set. The decision to update these is left the end user-- the task was not executed by the NM Server Installer nor the data_engine.

The following procedure describes how to migrate from old indexing to new indexing for the QoS sample data tables RN_QOS_DATA_x.

1. Stop the data_engine and any other activity against the NIS database (by stopping the hub and robots working against it). This step is required to the speed up the migration of indexes
2. **Important:** Take a backup of database before executing the steps mentioned below.
3. Using SQL Management Studio or similar, access the NIS database and execute the following steps to check which RN_QOS_DATA-tables require re-indexing.

   Expand the RN_QOS_DATA_xxxx tables to verify the type of indexing they are currently using.

   - If they have one clustered index with the same name as the table, then they are using the old indexing.

   - If they have two or three indexes (*_Idx0, _Idx1* and *_IdxSDP* all prefixed by table name), then they are using new indexing.

4. For all tables found to be using old indexing, execute the following step:

```
exec spn_utl_Reindex_SampledataTables
@TableNamePattern='RN_QOS_DATA_%'
```

Note: Modify the parameter **@TableNamePattern** to match the set of tables which should be re-indexed.

You can either execute the re-indexing in one step as above or execute it multiple times where each iteration covers one or more RN-tables as in example below:

```
exec spn_utl_Reindex_SampledataTable
@TableName='RN_QOS_DATA_0001'
```

5. Start any components stopped in step 1.

Note that re-indexing increases the storage space requirement for the database. The increased requirement ranges from 25% to 50%, where the increase approaches 50% if starting with a deployment which has only the original clustered index.

Performing Analysis on QoS sample data tables

Executing the stored procedure spn_ins_NISCC performs a detailed analysis of the QoS sample data tables and will report any issues which should be resolved:

- Log onto your database server
- Execute the following commands to analyze tables

```
spn_ins_NISCC 'list','all'
```

- Execute following command the review result of analysis

```
select * from vwn_ins_NISCC
```

- If the analysis above revealed any issues, you can run the following commands in order to resolve the issues:

```
spn_ins_NISCC 'fix','all'
```

- To review what has been resolved, execute the command

```
select * from vwn_ins_NISCC
```

Each database vendor has a different implementation with respect to execution of stored procedures. See detailed steps below for each database.

Note that you can change the value for the second argument (default 'all') to any number ('10' for example) to reduce the number of issues which should be reported/fixed. This is useful in situations where you do not have time to fix all issues in one execution.

Note: It is strongly advised to stop the data_engine probe and all other activity against database while executing the spn_ins_NISCC with the fix-option. This can be accomplished by placing the Primary Hub and Robots working against it into maintenance mode.

SqlServer

```
declare @lEM varchar(max), @lRC int;
-- analyse sample data tables
exec spn_ins_NISCC 'list', 'all', 5, @lEM, @lRC;
-- review result of analysis
select * from vwn_ins_NISCC;
-- resolve any issues
exec spn_ins_NISCC 'fix', 'all', 5, @lEM, @lRC;
-- review result of resolving issues
select * from vwn_ins_NISCC;
```

Oracle

```
ALTER SESSION SET NLS_SORT=BINARY_CI;
ALTER SESSION SET NLS_COMP=LINGUISTIC;
/
set serveroutput on;
/
-- analyse sample data tables
declare  lRC number; lEM varchar2(255);
begin
       spn_ins_NISCC('list', 'all', 5, lEM,lRC);
       dbms_output.put_line('lRC=' || lRC || ', lEM' || lEM);
end;
/
-- review result of analysis
select * from vwn_ins_NISCC;
-- resolve any issues
```

```
declare  lRC number; lEM varchar2(255);
begin
       spn_ins_NISCC('fix', 'all', 5, lEM,lRC);
       dbms_output.put_line('lRC=' || lRC || ', lEM' || lEM);
end;
/
-- review result of resolving issues
select * from vwn_ins_NISCC;
```

Mysql

```
-- analyze sample data tables
call spn_ins_NISCC('list','all', 5, @lEM,@lRC);
-- review result of analysis
select * from vwn_ins_NISCC;
-- resolve any issues
call spn_ins_NISCC('fix','all', 5, @lEM,@lRC);
-- review result of resolving issues
select * from vwn_ins_NISCC;
```

Appendix B: Advanced Indexing Topics

Advanced indexing when using report_engine/group_server

If using legacy report_engine or group_server, components, for example, when generating “on demand reports” from Dynamic Views, modifying the standard indexing of the RN_QOS_DATA-tables can improve database performance.

This section applies to MS SQL Server only.

Option One

RN-tables which have a very high number of distinct table_ids, as compared to the number of samples, will suffer performance degradation using standard indexing. A typical example is the RN-table for net_connect or interface_traffic QoS. For this combination the old (pre-NM Server v4.3x) clustered index over (table_id, sampletime) provided better performance compared to the non-clustered index over (table_id,sampletime) which replaced it (which generates an excessive amount of I/O).

The explanation is that report_engine's queries typically require non-indexed columns to qualify the rows required, something the clustered index handles fairly well. The remedy is to drop the current non-clustered index over (table_id,sampletime) and create a new non-clustered index over (table_id,sampletime,tz_offset, samplerate, samplevalue) as shown below:

```
create nonclustered index
   AllNClIdx
on
   rn_qos_data_0001_all(table_id,sampletime)
include
   (tz_offset,samplerate,samplevalue,samplemax)
```

Adding such an index to the designated RN-tables will reduce I/O to about the same level as the old clustered index. Please note that this index takes additional disk space--if you have data requiring 1 GB then the table with indexing will require a total of approximately 1.5 GB.

Option Two

A second approach to improving performance when generating “on demand reports” from Dynamic Views is to enable a covering index for RN_QOS_DATA_x tables (applies to MS SQL Server only).

For new QoS data

To enable a covering index for new RN_QOS_DATA tables, run this command:

```
update tbn_de_Config
      set ConfigValueNumeric = 0
where ConfigName = 'CreateMinimalTSIndex'
```

This will cause new RN tables to have Idx1 be a covering index, which enhances performance at the cost of disk space.

For existing QoS data

To enable covering indexes on existing RN_QOS_DATA-tables, run the following commands.

Note: The index update may take several hours, depending on database size.

For tables using the samplemax field:

```
declare @tName varchar(100)
declare @time varchar(30)

Declare tableHasMax CURSOR FOR
      select
          'RN_QOS_DATA_'     +     reverse(stuff('0000'      ,      1,
len(cast(s.qos_def_id  as  varchar(max))),  reverse(cast(s.qos_def_id
as  varchar(max)))))
      from S_QOS_DEFINITION s
      where hasmax = 1 and type = 0
      order by qos_def_id asc
OPEN tableHasMax
Fetch tableHasMax into @tName

WHILE @@FETCH_STATUS = 0
BEGIN

      set @time = convert(varchar(30), getdate(), 121);

      RAISERROR('%s %s starting', 0, 1, @time, @tname) WITH NOWAIT;

      exec spn_utl_IndexAdmin__DropIndex @TableName = @tName, @Target
= 'Idx1';

      exec spn_utl_IndexAdmin @mode='createindex', @IndexName='Idx1',
@IndexColumns='(sampletime,     table_id)     include     (samplerate,
samplevalue,   samplemax,   tz_offset)',   @IndexType='nonclustered',
@IndexOptions ='(fillfactor=75)', @TableNamePattern=@tName

      RAISERROR('%s %s done', 0, 1, @time, @tname) WITH NOWAIT;

      Fetch tableHasMax into @tName
      END
CLOSE  tableHasMax
DEALLOCATE tableHasMax
```

For tables without the samplemax field:

```
declare @tName varchar(100)
declare @time varchar(30)

Declare tableHasMax CURSOR FOR
      select
```

```
          'RN_QOS_DATA_'     +     reverse(stuff('0000'     ,     1,
len(cast(s.qos_def_id  as  varchar(max))),  reverse(cast(s.qos_def_id
as  varchar(max)))))
      from S_QOS_DEFINITION s
      where hasmax = 0 and type = 0
      order by qos_def_id asc
OPEN tableHasMax
Fetch tableHasMax into @tName

WHILE @@FETCH_STATUS = 0
BEGIN

      set @time = convert(varchar(30), getdate(), 121);

      RAISERROR('%s %s starting', 0, 1, @time, @tname) WITH NOWAIT;

      exec spn_utl_IndexAdmin__DropIndex @TableName = @tName, @Target
= 'Idx1';

      exec spn_utl_IndexAdmin @mode='createindex', @IndexName='Idx1',
@IndexColumns='(sampletime,     table_id)     include     (samplerate,
samplevalue,  tz_offset)',  @IndexType='nonclustered',  @IndexOptions
='(fillfactor=75)', @TableNamePattern=@tName

      RAISERROR('%s %s done', 0, 1, @time, @tname) WITH NOWAIT;

      Fetch tableHasMax into @tName
      END
CLOSE  tableHasMax
DEALLOCATE tableHasMax
```

Option Three

A third suggestion is to not use the DATEADD function on index columns. Here is an example report_engine query template that illustrates a potentially slow query:

```
SELECT
   AVG(samplevalue), STDEV(samplevalue), MIN(samplevalue), MAX(samplevalue),
MIN(samplerate)
FROM
   <rn_table> WITH (NOLOCK)
WHERE
   DATEADD(s,ISNULL(tz_offset,25200) - 25200,sampletime) between '<startdate>' and
'<enddate>'
and
   table_id = <table_id>
```

The first two options, which cover revising the index scheme, will allow the above example to run faster.

The alternative shown here is to not use the DATEADD function on index columns (an alternative is to force it to use Idx1). Here is a suggested revision to the above example query:

```
SELECT
   AVG(samplevalue), STDEV(samplevalue), MIN(samplevalue), MAX(samplevalue),
MIN(samplerate),COUNT(*)
FROM
```

```
   rn_qos_data_0001_xxx WITH (NOLOCK)
WHERE
   sampletime between '2010-10-08 09:00:00.000' and '2010-10-08 17:00:00.000'
and
   table_id = 2931842
```

Best Practices for Clustered and Non-Clustered Indexing

Best Practices for Clustered Indexes

Large amount of selects on a table, create a clustered index on the primary key of the table. Then create non-clustered indexes for all other columns used in selects and searches. Put non-clustered indexes on foreign key/primary key columns that are used in joins.

Best Practices for Non-clustered Indexes

Add non-clustered indexes for queries that return smaller result sets. Large results will have to read more table pages anyway so they will not benefit as much from a non-clustered index.

Add to columns used in WHERE clauses that return exact matches.

If a clustered index is not used on these columns, add an index for collections of distinct values that are commonly queried such as a first and last name column group.

Add for all columns grouped together for a given query that is expensive or very common on a large data table.

Add to foreign-key columns where joins are common that are not covered by the clustered index.

Appendix C: SQL Tools and Scripts

This section lists SQL code to perform the following tasks:

- Get_db_info_sqlserver
- get_index_fragmentation
- Fix_index_fragmentation
- Find_missing_nodes_in_dynamic_views
- DefragindexesBasedOnFragmentation (generic)
- Shrink_transaction_log_sqlserver
- Most costly unused indexes
- Top Costly Missing Indexes
- Indexes with the most contention
- Most logically fragmented Indexes
- Tables without clustered index
- Tables with primary key, without clustered index
- Objects with no indexes / Foreign keys that are not indexed
- Number of indexes per table
- Top SQL with highest CPU
- Top SQL with highest I/O
- Top SQL with highest Duration

Get db size info

This script provides a listing of data tables, sorted by size. Note that this is also included in the more general get info script that follows.

```
-- where is the data
CREATE TABLE #temp (
table_name sysname ,
row_count INT,
reserved_size VARCHAR(50),
data_size VARCHAR(50),
index_size VARCHAR(50),
unused_size VARCHAR(50))
SET NOCOUNT ON
INSERT #temp
EXEC sp_msforeachtable 'sp_spaceused ''?'''
SELECT a.table_name,
a.row_count,
COUNT(*) AS col_count,
a.data_size
FROM #temp a
INNER JOIN information_schema.columns b
ON a.table_name collate database_default
= b.table_name collate database_default
GROUP BY a.table_name, a.row_count, a.data_size
ORDER BY CAST(REPLACE(a.data_size, ' KB', '') AS integer) DESC
DROP TABLE #temp
go
```

Get_db_info_sqlserver

This script provides a summary of the NIS deployment size, what is being monitored, probe versions, frequently used probes, database size, data location, and information on specific data tables.

```
-- use NimsoftSLM -- <-- update this with your db name
-- go

-- summary of deployment size
select '1. # qos definitions' as item, COUNT(*) as cnt from
S_QOS_DEFINITION
union
select '2. # qos objects', COUNT(*) from S_QOS_DATA
union
select '3. # robots', COUNT(*) from CM_NIMBUS_ROBOT where is_hub = 0
and alive_time > DATEADD(hh, -1, getdate())
union
select '4. # hubs', COUNT(*) from CM_NIMBUS_ROBOT where is_hub = 1
and alive_time > DATEADD(hh, -1, getdate())
union
select '5. # computer systems', COUNT(*) from CM_COMPUTER_SYSTEM
where alive_time > DATEADD(hh, -1, getdate());

-- whats being monitored
select probe, COUNT(distinct qos) as QOS, COUNT(distinct source) as
#sources, COUNT(distinct target) as #targets
from S_QOS_DATA
group by probe
order by #targets desc, QOS desc

-- is everything running the same versions?
select probe_name, pkg_version, COUNT(*) as cnt from CM_NIMBUS_PROBE
where active = 1
and probe_name in ('controller', 'hub')
group by probe_name, pkg_version;
go

-- most frequently used probes
select probe_name, count(*) as Cnt
from CM_NIMBUS_PROBE
group by probe_name
order by Cnt desc
go

-- how big is our database
sp_spaceused

go

EXEC sp_helpindex 'RN_QOS_DATA_0001'

go

-- where is the data
CREATE TABLE #temp (
table_name sysname ,
row_count INT,
```

```
reserved_size VARCHAR(50),
data_size VARCHAR(50),
index_size VARCHAR(50),
unused_size VARCHAR(50))
SET NOCOUNT ON
INSERT #temp
EXEC sp_msforeachtable 'sp_spaceused ''?'''
SELECT a.table_name,
a.row_count,
COUNT(*) AS col_count,
a.data_size
FROM #temp a
INNER JOIN information_schema.columns b
ON a.table_name collate database_default
= b.table_name collate database_default
GROUP BY a.table_name, a.row_count, a.data_size
ORDER BY CAST(REPLACE(a.data_size, ' KB', '') AS integer) DESC
DROP TABLE #temp
go


-- get info on a specific table
-- sp_help RN_QOS_DATA_0008
```

get_index_fragmentation

```
USE NimsoftSLM
GO

print DB_NAME();

-- get fragmentation
SELECT object_name(IPS.object_id) AS [TableName],
   SI.name AS [IndexName],
   IPS.avg_fragmentation_in_percent,
   IPS.Index_type_desc,
   ps.row_count,
   IPS.avg_fragment_size_in_pages,
   IPS.avg_page_space_used_in_percent,
   IPS.record_count,
   IPS.ghost_record_count,
   IPS.fragment_count,
   IPS.avg_fragment_size_in_pages
FROM  sys.dm_db_index_physical_stats(db_id(DB_NAME()),  NULL,  NULL,
NULL , 'LIMITED') IPS
--  FROM  sys.dm_db_index_physical_stats(db_id(N'NimsoftSLM-NMStopo'),
NULL, NULL, NULL , 'LIMITED') IPS
INNER JOIN sys.tables ST WITH (nolock)
   ON IPS.object_id = ST.object_id
INNER JOIN sys.indexes SI WITH (nolock)
   ON IPS.object_id = SI.object_id
   AND IPS.index_id = SI.index_id
INNER JOIN sys.dm_db_partition_stats ps
   on IPS.object_id = ps.object_id
```

```
   AND ps.index_id = IPS.index_id
WHERE ST.is_ms_shipped = 0
ORDER BY IPS.avg_fragmentation_in_percent desc,  SI.name desc
GO
```

fix_index_fragmentation

```
msdb..[DBA_RebuildIndexesBasedOnFragmentation]         @maxfrag=30.0,
@maxdensity=90.0, @databasename= 'NimsoftSLM', @rebuild=1
```

find_missing_nodes_in_dynamic_views

```
-- # of origins not matching
select cs.origin as cmOrigin,  d.origin as sqdOrigin , count(*) from
CM_COMPUTER_SYSTEM cs inner join CM_NIMBUS_ROBOT r on cs.ip = r.ip
and cs.origin = r.origin inner join S_QOS_DATA d on cs.ip = d.host
where d.origin <> r.origin group by cs.origin, d.origin order by
count(*) desc


-- query to see if origins match up between CM_COMPUTER_SYSTEM,
CM_NIMBUS_ROBOT, S_QOS_DATA select cs.origin as cmOrigin,  r.origin
as robotOrigin, d.origin as sqdOrigin ,
len(cs.origin) as cmOriginLen,   len(r.origin) as robotOriginLen,
len(d.origin) as sqdOriginLen ,
*
from CM_COMPUTER_SYSTEM cs
inner join CM_NIMBUS_ROBOT r
on cs.ip = r.ip
and cs.origin = r.origin
inner join S_QOS_DATA d
on cs.ip = d.host
where d.origin <> r.origin

-- query to see if origins match up between CM_COMPUTER_SYSTEM and
CM_NIMBUS_ROBOT
select       cs.origin  as  cmOrigin,  r.origin  as  robotOrigin,
len(r.origin) as robotOriginLen, len(cs.origin) as cmOriginLen ,*
from CM_COMPUTER_SYSTEM cs inner join CM_NIMBUS_ROBOT r on cs.ip =
r.ip where cs.origin <> r.origin

-- query to update origins where S_QOS_DATA doesn't match the others
begin transaction
   update d set d.origin = r.origin
   from S_QOS_DATA d
   inner join CM_COMPUTER_SYSTEM cs
   on cs.ip = d.host
   inner join CM_NIMBUS_ROBOT r
   on cs.ip = r.ip
   and cs.origin = r.origin
   where d.origin <> r.origin
commit transaction
```

```
-- looking for the device from S_QOS_DATA
select * from S_QOS_DATA d
left join CM_CONFIGURATION_ITEM_METRIC m
on d.ci_metric_id = m.ci_metric_id
left join CM_CONFIGURATION_ITEM i
on m.ci_id = i.ci_id
left join CM_DEVICE c
on i.dev_id = c.dev_id
where d.probe = 'cdm' and
d.robot = ''

-- device info
select * from CM_DEVICE  d
where d.cs_id = ''
      or d.dev_id = ''

-- looking for the device from CM_COMPUTER_SYSTEM
select * From CM_COMPUTER_SYSTEM s
left join CM_GROUP_MEMBER cm
on s.cs_id = cm.cs_id
left join CM_GROUP cg
on cg.grp_id = cm.grp_id
where s.ip = ''
or s.name = ''
or s.cs_id = ''

-- all left joins to see where things break down
select distinct
c.dev_id,
r.address nimbus_address,
r.ip robotip,
r.domain,
r.hub hubname,
s.name robotname,
cg.name groupname,
s.nimbus_type,
d.source source,
d.origin,
s.os_type os_major,
s.os_name os_minor,
s.os_version,
s.os_description,
d.ci_metric_id,
d.qos,
d.target,
d.r_table,
d.probe,
d.table_id,
d.samplevalue value
from S_QOS_DATA d
left join CM_CONFIGURATION_ITEM_METRIC m
      on m.ci_metric_id=d.ci_metric_id
left join CM_CONFIGURATION_ITEM i
      on i.ci_id = m.ci_id
left join CM_DEVICE c
      on c.dev_id = i.dev_id
```

```
   left join CM_COMPUTER_SYSTEM s
         on c.cs_id = s.cs_id
   left join CM_GROUP_MEMBER cm
         on c.cs_id = cm.cs_id
   left join CM_GROUP cg
         on cg.grp_id = cm.grp_id
   left join CM_NIMBUS_ROBOT r
         on s.ip = r.ip and
            r.origin = d.origin
   where
         d.probe  = 'cdm'
         -- and d.robot = ''
         -- and d.origin = ''


-- query used by dynamic views to build the tree nodes
  select distinct
   c.dev_id,
   r.address nimbus_address,
   r.ip robotip,
   r.domain,
   r.hub hubname,
   s.name robotname,
   cg.name groupname,
   s.nimbus_type,
   d.source source,
   d.origin,
   s.os_type os_major,
   s.os_name os_minor,
   s.os_version,
   s.os_description,
   d.ci_metric_id,
   d.qos,
   d.target,
   d.r_table,
   d.probe,
   d.table_id,
   d.samplevalue value
   from S_QOS_DATA d,
   CM_CONFIGURATION_ITEM_METRIC m,
   CM_CONFIGURATION_ITEM i,
   CM_DEVICE c,
   CM_COMPUTER_SYSTEM s,
   CM_GROUP_MEMBER cm,
   CM_GROUP cg,
   CM_NIMBUS_ROBOT r
   where
    m.ci_metric_id=d.ci_metric_id and
    i.ci_id = m.ci_id and
    c.dev_id = i.dev_id and
    c.cs_id = s.cs_id and
    c.cs_id = cm.cs_id and
    cg.grp_id = cm.grp_id and
   s.ip = r.ip and
    r.origin = d.origin and
   d.probe  = 'cdm'
   UNION
     select distinct
```

```
   c.dev_id,
   r.address nimbus_address,
   r.ip robotip,
   r.domain,
   r.hub hubname,
   s.name robotname,
   cg.name groupname,
   s.nimbus_type,
   d.source source,
   d.origin,
   s.os_type os_major,
   s.os_name os_minor,
   s.os_version,
   s.os_description,
   d.ci_metric_id,
   d.qos,
   d.target,
   d.r_table,
   d.probe,
   d.table_id,
   d.samplevalue value
   from S_QOS_DATA d,
   CM_CONFIGURATION_ITEM_METRIC m,
   CM_CONFIGURATION_ITEM i,
   CM_DEVICE c,
   CM_COMPUTER_SYSTEM s,
   CM_GROUP_MEMBER cm,
   CM_GROUP cg,
   CM_NIMBUS_ROBOT r
   where
    m.ci_metric_id=d.ci_metric_id and
    i.ci_id = m.ci_id and
    c.dev_id = i.dev_id and
    c.cs_id = s.cs_id and
    c.cs_id = cm.cs_id and
    cg.grp_id = cm.grp_id and
    r.origin = d.origin and
   d.probe  = 'RSP'
```

sp_Generic_DefragindexesBasedOnFragmentation

```
USE [msdb]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE   PROCEDURE [dbo].[DBA_RebuildIndexesBasedOnFragmentation]

(
     @maxfrag float = 15.0           -- Maximum level of acceptable
fragmentation
```

```
   , @maxdensity float = 75.0        -- Minimum level of acceptable
density
   , @databasename NVARCHAR(512)   -- Database name to process , only
one database at a time
   , @rebuild bit = 0                  -- Define if we can rebuild
indexes. If we don't have ENT edition then we may need our data to
still available
   , @debug bit = 0                 -- Don't maintain indexes , show
me the queries.
)

AS

SET NOCOUNT ON ;

DECLARE @schemaname sysname;             -- Table schema

DECLARE @objectname nvarchar(512);       -- Table/view name

DECLARE @indexname nvarchar(512);        -- Index name

DECLARE @indexid INT;                    -- Index Id

DECLARE @currentfrag float;              -- Index fragmentation level
to compare against the defined threshold

DECLARE @currentdensity float;             -- Index Density level to
compare against the defined threshold

DECLARE @partitionnum VARCHAR(10);       -- Partition number (always
1 if index is NOT partitioned)

DECLARE @partitioncount bigint;          -- Partitions count (always
1 if index is NOT partitioned)

DECLARE @indextype VARCHAR(18);          -- Index type  (clustered ,
Non-clustered , XML ,..etc). Used for logging

DECLARE  @updatecommand  NVARCHAR(MAX);        --  used  to  hold  an
intermediate query

DECLARE  @command  NVARCHAR(MAX);                -- used to hold an
intermediate query

DECLARE @page_locks INT;                 -- Is Page_locks is enabled
on index ?

DECLARE @has_lob INT;                    -- Does table has any LOBs ?

DECLARE @db_id INT;                      -- Database ID

DECLARE @object_type CHAR(1);              -- Object type (U= user
table , V = View)

DECLARE  @server_edition  bit;                   -- instance edition
(1=Enterprise , 0= other) to determine if index can be rebuilt ONLINE

/*
```

```
We will be creating a table in MSDB to keep track of maintained
indexes.

This can help in tracking which indexes were maintains , time it took
, fragmentation level , ...etc

*/

IF (SELECT OBJECT_ID('msdb..index_maintenance_log')) IS NULL
   BEGIN
   CREATE TABLE msdb..index_maintenance_log
       (
        Db NVARCHAR(512) NOT NULL
       ,idx NVARCHAR(512) NOT NULL
       ,IndexType VARCHAR (64) NOT NULL
       ,currentfrag float NOT NULL
       ,time_ datetime NOT NULL DEFAULT GETDATE()    -- Time when the
index maintenance was done
       ,notes VARCHAR(450) NULL
       )  ON [PRIMARY]
   END


/*
Create a temp table to hold indexes information (table , index
fragmentation , index name , ...etc to later loop on them)
*/

CREATE TABLE #work_to_do(
     IndexID INT NOT NULL
   , IndexName NVARCHAR(512) NULL
   , TableName NVARCHAR(512) NULL
   , Tableid INT NOT NULL
   , SchemaName VARCHAR(255) NULL
   , IndexType VARCHAR(18) NOT NULL
   , PartitionNumber INT NOT NULL
   , PartitionCount INT NULL
   , CurrentDensity float NOT NULL
   , CurrentFragmentation float NOT NULL
   , page_locks INT NULL
   , Has_lob INT NOT NULL DEFAULT 0
   , object_type VARCHAR(1)
   , Index_disabled bit NOT NULL DEFAULT 0  -- Is index disabled ?
)


IF NOT exists (SELECT name FROM master.sys.databases WHERE name =
@databasename AND state_desc = 'Online' AND  is_read_only =0 )
BEGIN
EXEC (' Raiserror (''This database doesn''t exist OR indexes can''t
be updated : '+@databasename +''',10,1)')
RETURN;
END
-- Get server edition to see if Online can be used
SELECT @server_edition = CASE WHEN CAST (SERVERPROPERTY ('edition')
AS NVARCHAR(200)) LIKE '%enterprise%' THEN 1 ELSE 0 END
--- Log the start of work against database
```

```
if @debug =0
begin
INSERT        INTO        msdb..index_maintenance_log        (Db,idx
,IndexType,currentfrag , notes)
VALUES (@databasename , 'NONE' ,'NONE', 0.0 ,'Db_start')
end
-- Get the tables , indexes info beside fragmentation level
-- We will be skipping small tables (< 512 Pages)
INSERT INTO #work_to_do(
   IndexID,  Tableid,    IndexType,  PartitionNumber,  CurrentDensity,
CurrentFragmentation
   )
   SELECT
       fi.index_id
       , fi.OBJECT_ID
       , fi.index_type_desc AS IndexType
       , CAST(fi.partition_number AS VARCHAR(10)) AS PartitionNumber
       , fi.avg_page_space_used_in_percent AS CurrentDensity
       , fi.avg_fragmentation_in_percent AS CurrentFragmentation
   FROM   sys.dm_db_index_physical_stats(DB_ID(@databasename),   NULL,
NULL, NULL, 'SAMPLED') AS fi
   WHERE       (fi.avg_fragmentation_in_percent   >=   @maxfrag     OR
fi.avg_page_space_used_in_percent < @maxdensity)
        AND    page_count > 512
        -- Skip heaps
   AND fi.index_id > 0
/*
 Because dm_db_index_physical_stats doesn't include all info we need
, we will join the result with some catalog views
to  assign  the  index  names,  schema  names,  table  names  ,partition
counts and other information
*/

SET @updatecommand =
'UPDATE  #work_to_do  SET  TableName  =  o.name,  SchemaName  =  s.name,
IndexName          =          i.Name          ,page_locks          =
i.allow_page_locks,object_type=o.type ,Index_disabled = i.is_disabled
,PartitionCount = (SELECT COUNT(*) pcount
FROM '
   + QUOTENAME(@databasename) + '.sys.Partitions p
where  p.Object_id = w.Tableid
AND p.index_id = w.Indexid)
,Has_lob=isnull   ((select   top   1   col.user_type_id   from   '+
QUOTENAME(@databasename)+'.sys.columns   col   INNER   JOIN   '   +
QUOTENAME(@databasename)  +  '.sys.types  ty  ON  col.user_type_id  =
ty.user_type_id   WHERE   col.object_id=w.Tableid        AND   (ty.name
IN(''xml'',''image'',''text'',''ntext'')          OR          (ty.name
IN(''varchar'',''nvarchar'',''varbinary'') AND col.max_length = -1)))
,0)
FROM '
   + QUOTENAME(@databasename) + '.sys.objects o INNER JOIN '
   +  QUOTENAME(@databasename)  +  '.sys.schemas  s  ON  o.schema_id  =
s.schema_id
INNER JOIN #work_to_do w ON o.object_id = w.tableid INNER JOIN '
   +  QUOTENAME(@databasename)  +  '.sys.indexes  i  ON  w.tableid  =
i.object_id and w.indexid = i.index_id
';
   --print @updatecommand
```

```
BEGIN Try
   EXEC(@updatecommand)
END try

-- Log the error to SQL server log
BEGIN Catch
SELECT @updatecommand = 'Smart indexing error : ' + @databasename + '
: ' + ERROR_MESSAGE ( )
   RAISERROR (@updatecommand, 16, 1) WITH LOG
END catch
/*
Declare a cursor to loop over all indexes that have fragmentation
level more than threshold or page density less than threshold
*/

DECLARE rebuildindex CURSOR FOR
   SELECT
           object_type
           , has_lob
           , page_locks
           , QUOTENAME(IndexName) AS IndexName
           , TableName
           , SchemaName
           , IndexType
           , PartitionNumber
           , PartitionCount
           , CurrentDensity
           , CurrentFragmentation
   FROM    #work_to_do i
        WHERE     Index_disabled = 0   -- Indexes processes must be
enabled
   ORDER BY TableName, IndexID;

-- Open the cursor.
OPEN rebuildindex;
-- Loop through the tables, indexes and partitions.
FETCH NEXT
   FROM rebuildindex
   INTO   @object_type,@has_lob,@page_locks,@indexname,   @objectname,
@schemaname,      @indextype,      @partitionnum,      @partitioncount,
@currentdensity, @currentfrag;
WHILE @@FETCH_STATUS = 0
BEGIN
-- We will process indexes only with Page_locks enabled
IF @page_locks = 1
BEGIN
        -- Initial block that applied to all cases
 SELECT  @command  =  'ALTER  INDEX  '  +  @indexname  +  '  ON  '  +
QUOTENAME(@databasename)  +'.'  +  QUOTENAME(@schemaname)  +  '.'  +
QUOTENAME(@objectname);
   -- If the index is >=30 fragmented ad issue and @rebuild parameter
= 1 then REBUILD Otherwise REORGANIZE.
        -- We will defragment indexes on views
   IF @currentfrag >= 30 AND @object_type <> 'v'
               BEGIN;
               SELECT @command = @command + CASE @Rebuild WHEN 1 THEN
' REBUILD ' ELSE ' REORGANIZE ' END ;
```

```
   -- We can REORGANIZE a single index partition but cannot REBUILD a
single index partition ONLINE.
        -- Also , we can't rebuild an index online if it has LOBs
(Varchar(max) , nvarchar(max) , text ....etc)

               IF  @has_lob  =  0  AND  @partitioncount  <=  1  AND
@server_edition = 1 AND @Rebuild = 1
               SELECT @command = @command + ' WITH(ONLINE = ON) ';
                                -- Index is partitioned , define the
partition number
               IF @partitioncount > 1
                   SELECT  @command  =  @command  +  '  PARTITION='  +
@partitionnum;
                               -- Index will be reorganized , we will
compact the LOBs
               IF @Rebuild = 0
                   SELECT   @command   =   @command   +   '   WITH   (
LOB_COMPACTION = ON )'
               END;
IF @currentfrag < 30 OR  @object_type ='v'
               BEGIN;
               SELECT @command = @command + ' REORGANIZE ';
               IF @partitioncount > 1
                   SELECT  @command  =  @command  +  '  PARTITION='  +
@partitionnum ;
               SELECT @command = @command + ' WITH ( LOB_COMPACTION =
ON )'
               END;
       -- @debug= 1 ; We need to see the T-SQL command only , not
running it
       IF @debug = 1
       PRINT @command
       ELSE
           BEGIN
               BEGIN Try
                   EXEC (@command);
                -- Log the index operation
               If @debug =0
               Begin
                   INSERT  INTO  msdb..index_maintenance_log  (Db,idx
,IndexType,currentfrag )
                   VALUES  (@databasename  ,  QUOTENAME(@schemaname)  +
'.'    +    QUOTENAME(@objectname)    +    '.'    +    @indexname
,@indextype,@currentfrag)
               End
              END Try
               BEGIN Catch
               SELECT @command = 'Smart indexing error : ' + @command
+ ' : ' + ERROR_MESSAGE ( )
               RAISERROR (@command, 16, 1) WITH LOG
               END Catch
           END
END

-- @page_locks = 0

ELSE
```

```
BEGIN

INSERT        INTO        msdb..index_maintenance_log        (Db,idx
,IndexType,currentfrag , notes)
VALUES    (@databasename    ,    QUOTENAME(@schemaname)    +    '.'    +
QUOTENAME(@objectname) + '.' + @indexname ,@indextype,@currentfrag
,'page locks are disabled')
END

      FETCH        NEXT        FROM        rebuildindex        INTO
@object_type,@has_lob,@page_locks,@indexname,            @objectname,
@schemaname,     @indextype,     @partitionnum,     @partitioncount,
@currentdensity, @currentfrag;

END;

-- Close and deallocate the cursor.

CLOSE rebuildindex;

DEALLOCATE rebuildindex;


-- Log the End of work against database
If @debug =0
Begin
INSERT        INTO        msdb..index_maintenance_log        (Db,idx
,IndexType,currentfrag , notes)
VALUES (@databasename , 'NONE' ,'NONE', 0.0 ,'Db_End')
End

-- Clean the log table (11+ days old)

Delete FROM msdb..index_maintenance_log WHERE time_ < GETDATE() - 10
```

shrink_transaction_log_sqlserver

-- Useful links: http://msdn.microsoft.com/en-us/library/ms178037.aspx

```
-- MAKE SURE RECOVERY MODE IS SET TO SIMPLE
--
--
ALTER DATABASE [NimsoftSLM] SET RECOVERY SIMPLE WITH NO_WAIT

-- see whats available (or full)
DBCC SQLPERF(LOGSPACE);

-- backup
backup log NimsoftSLM with truncate_only

-- shrink
DBCC SHRINKFILE (NimsoftSLM_log, 1);
```

Most costly unused indexes

```
-- Create required table structure only.
-- Note: this SQL must be the same as in the Database loop given in
the following step.
SELECT TOP 1
        DatabaseName = DB_NAME()
        ,TableName = OBJECT_NAME(s.[object_id])
        ,IndexName = i.name
        ,user_updates
        ,system_updates
        -- Useful fields below:
        --, *
INTO #TempUnusedIndexes
FROM   sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON  s.[object_id] = i.[object_id]
    AND s.index_id = i.index_id
WHERE  s.database_id = DB_ID()
    AND OBJECTPROPERTY(s.[object_id], 'IsMsShipped') = 0
    AND    user_seeks = 0
    AND user_scans = 0
    AND user_lookups = 0
    AND s.[object_id] = -999  -- Dummy value to get table structure.
;

-- Loop around all the databases on the server.
EXEC sp_MSForEachDB    'USE [?];
-- Table already exists.
INSERT INTO #TempUnusedIndexes
SELECT TOP 50
        DatabaseName = DB_NAME()
        ,TableName = OBJECT_NAME(s.[object_id])
        ,IndexName = i.name
        ,user_updates
        ,system_updates
FROM   sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON  s.[object_id] = i.[object_id]
    AND s.index_id = i.index_id
WHERE  s.database_id = DB_ID()
    AND OBJECTPROPERTY(s.[object_id], ''IsMsShipped'') = 0
    AND    user_seeks = 0
    AND user_scans = 0
    AND user_lookups = 0
    AND i.name IS NOT NULL    -- Ignore HEAP indexes.
ORDER BY user_updates DESC;'
-- Select records.
SELECT TOP 30 * FROM #TempUnusedIndexes ORDER BY [user_updates] DESC
-- Tidy up.
DROP TABLE #TempUnusedIndexes
```

Top Costly Missing Indexes

```
SELECT  TOP 30
```

```
        [Total Cost]  = ROUND(avg_total_user_cost * avg_user_impact *
(user_seeks + user_scans),0)
        , avg_user_impact
        , TableName = statement
        , [EqualityUsage] = equality_columns
        , [InequalityUsage] = inequality_columns
        , [Include Cloumns] = included_columns
FROM        sys.dm_db_missing_index_groups g
INNER JOIN    sys.dm_db_missing_index_group_stats s
       ON s.group_handle = g.index_group_handle
INNER JOIN    sys.dm_db_missing_index_details d
       ON d.index_handle = g.index_handle
ORDER BY [Total Cost] DESC;
```

Indexes with the most contention

```
Select top 50 dbid=database_id, objectname=object_name(s.object_id)
   , indexname=i.name, i.index_id      --, partition_number
   , row_lock_count, row_lock_wait_count
   ,   [block   %]=cast   (100.0   *   row_lock_wait_count   /   (1   +
row_lock_count) as numeric(15,2))
   , row_lock_wait_in_ms
   , [avg row lock waits in ms]=cast (1.0 * row_lock_wait_in_ms / (1
+ row_lock_wait_count) as numeric(15,2))
into #tmp
from sys.dm_db_index_operational_stats (1, NULL, NULL, NULL) s
   ,sys.indexes i
where 1=2

EXEC sp_MSForEachDB 'use [?]; declare @dbid int; set @dbid = db_id();
insert into #tmp
Select top 20 dbid=database_id, objectname=object_name(s.object_id)
   , indexname=i.name, i.index_id      --, partition_number
   , row_lock_count, row_lock_wait_count
   ,   [block   %]=cast   (100.0   *   row_lock_wait_count   /   (1   +
row_lock_count) as numeric(15,2))
   , row_lock_wait_in_ms
   , [avg row lock waits in ms]=cast (1.0 * row_lock_wait_in_ms / (1
+ row_lock_wait_count) as numeric(15,2))
from sys.dm_db_index_operational_stats (@dbid, NULL, NULL, NULL) s
   ,sys.indexes i
where objectproperty(s.object_id,''IsUserTable'') = 1
and i.object_id = s.object_id
and i.index_id = s.index_id
order by row_lock_wait_count desc'

select top 30 db_name(dbid) as DBName,*
from #tmp where dbid > 4 and row_lock_wait_count <> 0
order by row_lock_wait_count desc
```

Most logically fragmented Indexes

```
SELECT TOP 30
       sc.name + '.' + OBJECT_NAME(s.[object_id]) 'table' , o.type
```

```
        ,IndexName = i.name
        ,[Fragmentation %] =
avg_fragmentation_in_percent,i.allow_page_locks,sum(p.rows) as rowCnt
, page_count FROM sys.dm_db_index_physical_stats(db_id(),null, null,
null,  null)  s  INNER  JOIN  sys.indexes  i  ON  s.[object_id]  =
i.[object_id]
    AND s.index_id = i.index_id

join sys.partitions p
on i.object_id = p.object_id
and i.index_id = p.index_id

join sys.objects o
on i.object_id = o.object_id

join sys.schemas sc
on o.schema_id = sc.schema_id

WHERE s.database_id = DB_ID()
      AND i.name IS NOT NULL    -- Ignore HEAP indexes.
    AND OBJECTPROPERTY(s.[object_id], 'IsMsShipped') = 0

group  by  sc.name  +  '.'  +  OBJECT_NAME(s.[object_id])      ,
o.type,i.name,avg_fragmentation_in_percent,i.allow_page_locks,page_co
unt
ORDER BY [Fragmentation %] DESC
```

Tables without clustered index

```
SELECT    convert(varchar(200),NULL)  as DB,  OBJECT_NAME(OBJECT_ID)
as TableName,
convert(int,0) as RowNum
into #tmp
    FROM     SYS.INDEXES
    WHERE    1=2

exec     sp_msforeachdb     'use     [?];     if     ''?''     in
(''tempdb'',''master'',''model'',''msdb'') return;
insert into #tmp(DB,TableName,RowNum)
SELECT    ''?'',OBJECT_NAME(i.OBJECT_ID), sum(p.rows)
    FROM     SYS.INDEXES i
join sys.partitions p
   on i.object_id = p.object_id
   and i.index_id = p.index_id
    WHERE    i.INDEX_ID = 0
        AND OBJECTPROPERTY(i.OBJECT_ID,''IsUserTable'') = 1
    Group BY OBJECT_NAME(i.OBJECT_ID)'

select * from #tmp
where [tableName] not like 'spt_%'
order by Rownum desc
```

Tables with primary key, without clustered index

```
SELECT    convert(varchar(200),NULL)   as DB,   OBJECT_NAME(OBJECT_ID)
as TableName,
convert(int,0) as RowNum
into #tmp
    FROM     sys.indexes
    WHERE    1=2

exec    master..sp_msforeachdb    'use    [?];    if    ''?''    in
(''tempdb'',''master'',''model'',''msdb'') return;
insert into #tmp(DB,TableName,RowNum)
SELECT    ''?'',OBJECT_NAME(i.OBJECT_ID), sum(p.rows)
    FROM     sys.indexes i
join sys.partitions p
   on i.object_id = p.object_id
   and i.index_id = p.index_id
    WHERE    i.INDEX_ID = 0
        AND OBJECTPROPERTY(i.OBJECT_ID,''IsUserTable'') = 1
    and OBJECT_NAME(i.OBJECT_ID) not in

(
SELECT
   T.TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES AS T
WHERE NOT EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
      WHERE CONSTRAINT_TYPE = ''PRIMARY KEY''
         AND T.TABLE_NAME = TC.TABLE_NAME)
         AND T.TABLE_TYPE = ''BASE TABLE''
)
    Group BY OBJECT_NAME(i.OBJECT_ID)'

select * from #tmp
where [tableName] not like 'spt_%'
order by Rownum desc
```

Objects with no indexes / Foreign keys that are not indexed

```
IF OBJECT_ID('tempdb..#t1') IS NOT NULL DROP TABLE #t1

IF OBJECT_ID('tempdb..#FKTable') IS NOT NULL DROP TABLE #FKTable

--Create index temp table
CREATE TABLE #t1
( do integer default(0),
index_name varchar(100),
index_descrip varchar(200),
index_keys varchar(200),
table_name varchar(100))

--Create FK temp table
```

```
CREATE TABLE #FKTable
( fk_name varchar(100),
fk_keys varchar(200),
fk_keyno int,
table_name varchar(100))

--Collect and uppdate all index info
EXEC  sp_msforeachtable  "insert  #t1  (index_name,  index_descrip,
index_keys) exec sp_helpindex '?'; update #t1 set table_name = '?',
do = 1 where do = 0"
UPDATE #t1 SET table_name = replace(table_name , '[', '')
UPDATE #t1 set table_name = replace(table_name , ']', '')


--Collect all index info
INSERT INTO #FKTable
SELECT  OBJECT_NAME(constid)  AS  FKName,  COL_NAME(fkeyid,  fkey)  AS
FKColumn, keyno,
s.name + '.' + OBJECT_NAME(fkeyid) AS TabName
FROM sysforeignkeys k
JOIN sys.objects c
ON k.constid = c.object_id
JOIN sys.schemas s
ON c.schema_id = s.schema_id

--If FK have two or more columns add them in one row to be able to
compare with index columns.

DECLARE @FKName AS VARCHAR(200), @FKColumn as VARCHAR(100)

DECLARE FKCurusor CURSOR FOR
SELECT  OBJECT_NAME(constid)  AS  FKName,  COL_NAME(fkeyid,  fkey)  AS
FKColumn
FROM sysforeignkeys k
JOIN sysobjects c
ON k.constid = c.id
WHERE keyno > 1
ORDER BY keyno

DELETE FROM #FKTable WHERE fk_keyno > 1

OPEN FKCurusor
FETCH NEXT FROM FKCurusor INTO @FKName,@FKColumn
WHILE (@@FETCH_STATUS = 0)
BEGIN

UPDATE #FKTable SET
fk_keys = fk_keys + ', ' + @FKColumn
WHERE fk_name = @FKName


FETCH NEXT FROM FKCurusor INTO @FKName,@FKColumn

END

CLOSE FKCurusor
DEALLOCATE FKCurusor
/*
```

```
SELECT * FROM #FKTable ORDER BY table_name

SELECT * FROM #t1 ORDER BY table_name
*/
PRINT '
--------------------------------------------------------------------------
FK MISSING Indexes
--------------------------------------------------------------------------
-------'

SELECT DISTINCT table_name, fk_name
FROM #FKTable f1
WHERE NOT EXISTS (
SELECT fk_name
FROM #FKTable f
INNER JOIN #t1 t
ON f.table_name = t.table_name
WHERE f1.fk_name = f.fk_name
AND fk_keys = index_keys)
```

Number of indexes per table

```
with x as
(select tablename, name, count(*) as cnt from #tmp2
group by tablename,name
)
select * from (
select tablename,count(cnt) as cnt2 from x
where tablename not like 'sys%' and name not like '_WA%'
group by tablename) as y
where cnt2 > 5 order by cnt2 desc
```

Top SQL with highest CPU

```
with x as (SELECT TOP 1000
 [Average CPU used] = total_worker_time / qs.execution_count
,[Total CPU used] = total_worker_time
,[Execution count] = qs.execution_count
,[Individual Query] = SUBSTRING (qt.text,qs.statement_start_offset/2,
         (CASE WHEN qs.statement_end_offset = -1
            THEN LEN(CONVERT(NVARCHAR(MAX), qt.text)) * 2
          ELSE qs.statement_end_offset END -
qs.statement_start_offset)/2)
,[Parent Query] = qt.text
,DatabaseName = DB_NAME(qt.dbid)
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
WHERE  total_worker_time / qs.execution_count> 200
ORDER BY [Total CPU used] DESC
)
select * from x order by [Execution count] desc
```

Top SQL with highest I/O

```
with x as (SELECT TOP 1000
 [Average  IO]  =  (total_logical_reads  +  total_logical_writes)  /
qs.execution_count
,[Total IO] = (total_logical_reads + total_logical_writes)
,[Execution count] = qs.execution_count
,[Individual Query] = SUBSTRING (qt.text,qs.statement_start_offset/2,
         (CASE WHEN qs.statement_end_offset = -1
            THEN LEN(CONVERT(NVARCHAR(MAX), qt.text)) * 2
          ELSE         qs.statement_end_offset         END         -
qs.statement_start_offset)/2)
        ,[Parent Query] = qt.text
,DatabaseName = DB_NAME(qt.dbid)
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
where     (total_logical_reads     +     total_logical_writes)     /
qs.execution_count > 1000
ORDER BY [Total IO] DESC)
select * from x order by [Execution count] desc
```

Top SQL with highest Duration

```
with x as (SELECT TOP 1000
[Total time] = total_elapsed_time,
[Average time] = total_elapsed_time / qs.execution_count
,[Execution count] = qs.execution_count
,[Individual Query] = SUBSTRING (qt.text,qs.statement_start_offset/2,
         (CASE WHEN qs.statement_end_offset = -1
            THEN LEN(CONVERT(NVARCHAR(MAX), qt.text)) * 2
          ELSE qs.statement_end_offset END -
qs.statement_start_offset)/2)
,[Parent Query] = qt.text
,DatabaseName = DB_NAME(qt.dbid),
creation_time, last_execution_time
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) as qt
where total_elapsed_time / qs.execution_count > 1000
ORDER BY [Total time] DESC)
select * from x order by [Execution count] desc
```

Appendix D: TNT2 Data Model