Infor LN Performance, Tracing and Tuning Guide for SQL Server

Publication Information

Release: Infor LN 10.x

Publication date: November 29, 2012

Document code: B0079B

Contents

About this guide

This document provides guidelines to improve the performance of an Infor LN environment on a SQL Server database by tracing and tuning the environment.

Note: This document is a comprehensive compilation; however there may be instances wherein relevant information or procedures may have been omitted. Therefore, we strongly recommend verifying the proposed changes in a test environment before moving to production. The information provided may not hold true for future versions of the SQL Server database.

Intended audience

This document is intended for intermediate to expert Infor LN and database Administrators and Technical Consultants and aims to better the performance of the Infor LN system.

Related documents

Certain sections in this document are described in more detail in other documents. The following documents help to extend the knowledge in particular areas.

- *Infor LN - Performance, Tracing and Tuning Guide (U9357 US)*
- *Infor LN - Sizing guide (B0045 US)*
- *Infor Enterprise Server - Technical Reference Guide for* Microsoft SQL Server *Database Driver (U8173 US)*

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in "Contacting Infor" on page 6.

A good reference to SQL Server documentation that helps for tracing and tuning is important. The following site has proven content: http://msdn.microsoft.com/en-us/library/ms130214.aspx

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at www.Infor.com/Inforxtreme.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@Infor.com.

Chapter 1 Tuning SQL Server for Infor LN

1

The performance of Infor LN depends highly on the database performance. This chapter guides you through the most important tuning areas of SQL Server.

Introduction

Tuning SQL Server is not the standard way to repair your system. This chapter describes what can be done based on the performance issues you have noticed.

Usually, Infor LN runs efficiently with an out-of-the box SQL Server database, as long as setup of the SQL Server database is planned appropriately. Additional tuning will further improve performance.

Regarding SQL Server Performance Tuning, it is important to note that performance tuning is not a search for the “magic” switch. Each area of tuning is very important to the overall performance, and must be approached using a logical methodology. Proper organization, logical changes, and documentation classify the difference between effective and ineffective tuning. Performance tuning along with capacity planning and sizing will allow you to design, setup, implement, and maintain a smooth and well operating system.

In this chapter, the term CPU is used frequently and refers to the Windows Operation systems description of this term. In practice, it can be a CPU core or CPU Hyperthread. Only when an exception applies, it is explicitly stated.

The following topics will be discussed in more detail to make the database “tuning” easier.

- SQL Server Information resources
- I/O Setup
- Storage Setup
- Network setup
- Database statistics
- (N)VARCHAR
- SQL Server Parameters

SQL Server Information resources

For Generic SQL Server Information and SQL Server Tuning, Information is available on the internet and from various Microsoft SQL Server books. However, most of them are copies of SQL Server Books Online; Hence, SQL Server Books Online (BOL) is one of the most valuable Information sources for SQL Server.

For more Information, you can refer to the following books (Series):

- “Inside Series (2005 & 2008 editions available) MSQL Server by Microsoft Press”.
- “Microsoft SQL Server 2008 Internals” by Microsoft Press
- “SQL Server 2008 Internals and Troubleshooting” by Wrox Press
- “Microsoft SQL Server (2000 & 2005 edition available) Performance optimization and Tuning handbook” by Ken England.
- “SQL Server Magazine”. This magazine keeps you Informed about Microsoft SQL Server on a monthly basis.
- SQLCAT website (SQLCAT). Website where SQL Server Developers places additional Information (not part of the official SQL Server build).

I/O Setup

The *Infor LN - Performance, Tracing and Tuning Guide (U9357)* provides guidelines for a correct I/O setup. For SQL Server, the following additional guidelines are important:

- Configure Stripe sizes for the SQL Server volumes to the recommended size of 64K or a value divisible by 8K.
- Configure NTFS Allocation Unit for the SQL Server used file systems to the recommended size of 64K or a value divisible by 8K.
- Always use new configured and formatted disks for SQL Server data and log storage. Most SQL Administrators tend to check only the internal SQL Server data and index fragmentation. It is also important to check regularly the existence of highly fragmented SQL Server file storage on Windows file system level.

Storage Setup

A good storage setup is important for SQL Server performance. The following guidelines will help design and implement the storage setup:

- Usage of filegroups to improve performance is not recommended. The best approach is to spread the I/O across multiple disks based on your requirement. Only “hotspots” (e.g. for a specific, heavily used, table as the device queue) in the Infor LN database that exist over time (it is always “hot”) are possible scenarios for I/O separation using Filegroups. The %BSE%/lib/msql/msql_storage_param fully supports this process.

The following example displays the correct setup of the %BSE%/lib/msql/msql_storage_param to separate a table:

```
ttaad320:000:T:group:0400:FILEGROUP HOTSPOT
*:*:T:group:0400:FILEGROUP PRIMARY
*:*:I:FILEGROUP PRIMARY
```

- Split the Infor Database in multiple data files. For CPU cores up to 8, it is recommended that you use one datafile per core. Above 8 CPU cores: use 8 datafiles and add an additional datafile for every two additional cores. The reason behind this (same applies for tempdb) that each data file begin with its own header page, PFS, SGAM and GAM page. Contention on these pages is reduced by creating multiple of these pages. This is done by creating multiple data files.
- I/O demand for the Infor LN Tempdb is very low in normal circumstances. Additional functionality that also includes the use of Infor LN databases increases the I/O demand of Tempdb; In such a scenario, separation of Tempdb to its own volume is recommended. High I/O demand in Tempdb is usually due to poorly written SQL or complex queries.
- Split the SQL Server Tempdb database into multiple data files. It is recommended that you use 1 datafile per 2 CPU cores to a maximum of 8 data files.
- The Infor LN database must be created with the appropriate size. For example, if at least 150 GB is needed within a year, create it with this size from the beginning. Do not let it grow to the required size and cause additional fragmentation in the database and the NTFS volume.
- Set autogrowth to a minimum of 10 percent or 1 GB (for each extent) to avoid fragmentation on file system level. Never use the default extent size of 1 MB!
- SQL Server log is not circular by nature, like Oracle, but keeps growing. Setup a good Log backup procedure to free up log space and keep the log within reasonable values.

For more Information on I/O and Storage Setup best practices, refer to the SQLCAT website (SQLCAT) and the Microsoft knowledge base.

Network Setup

2-Tier

For optimal SQL Server (network) performance, use the following guidelines for porting sets 8.7a onwards:

- The Infor LN SQL Server database driver uses the SNAC Driver (SQL Native Access Client) instead of WDAC/MDAC. As a result the Infor LN database driver accesses SQL Server via Shared Memory instead of Named Pipes. It delivers an approximately 10 percent better performance when compared to NP protocol in a 2-Tier configuration. Shared memory is always enabled on a SQL Server installation. When not specified, the SNAC enabled Infor LN database driver always uses shared memory. The following example in $BSE/lib/defaults/db_resource enables shared memory when it is not specified:

```
msql_serverhost=<systemname>
```

For optimal SQL Server performance for porting sets **prior** to 8.7a, use the following guidelines:

- Named Pipes (np) is the preferred protocol when SQL Server and Infor LN are installed on the same server (2-Tier configuration). It delivers an approximately 20 percent better performance when compared to TCP/IP protocol in a 2-Tier configuration. Network protocols can be configured with the network configuration utilities of SQL Server by switching protocols ON/OFF and giving their order. However, this does not guarantee that the right protocol is used. Using the Infor LN resource variable in %BSE%/lib/defaults/db_resource will always give the correct protocol:

```
msql_serverhost=np:<systemname>
```

3-Tier

TCP/IP is the preferred protocol when SQL Server and Infor LN are installed on separate servers (3-Tier configuration). This is for both SNAC enabled SQL Server database drivers (porting set 8.7a and beyond) and the older WDAC/MDAC based database drivers.

Infor LN with SQL Server uses relatively small network packages for communication. Actual package size is around 300-500 bps. Effectively, the bandwidth size of a LAN connection is not a plausible limitation, but the amount of packages, is. For a single 1 Gbps network adapter, 25,000 – 30,000 network packets are the upper limit.

Database Statistics

Creating database statistics on SQL Server is important. In most circumstances the “auto create statistics” database option creates the required statistics. When it does not, it is recommended that you to create statistics manual as well. Create missing statistics on newly created data at regular intervals. You must create and update statistics when there is low activity on the system.

The following procedure creates missing statistics in a particular database:

```
sp_createstats @fullscan = ‘fullscan’, @indexonly = ‘indexonly’
```

This procedure scans all the data and statistics created only for index. In SQL Server terminology, index means statistics for key distribution of the index i.e. the first column of an index key.

The next example creates missing statistics for all columns, including columns in a composite index key, other than the first. This approach is recommended for creating statistics for SQL Server. However, there is one drawback. More storage for statistics is needed, but in most cases the SQL Server Query optimizer can create a better optimal plan:

```
sp_createstats @fullscan = ‘fullscan’
```

Update the SQL Server database statistics at regular intervals, when there is low activity on the system. The following procedure updates statistics in a particular database:

```
sp_updatestats
```

(N)VARCHAR

From porting set 8.4b, the Infor LN SQL Server database driver uses VARCHAR and NVARCHAR data types as standard. This will significantly reduce data size and data growth. As a result the amount of I/O reduces and overall performance will improve as well.

SQL Server parameters

In most cases, Infor LN runs efficiently with an out-of-the box SQL Server database with standard parameters. Modifications of parameters can eliminate a few issues or influence resource usage on high-end SQL Server systems. The parameters used by benchmarks and customers are listed in this table.

sp_configure parameter	**recommended value**	**importance**
Affinity (64) I/O Mask	Default (0)	Optional (High-end only)
Affinity (64) mask	Default (0)	Optional (Special purpose)
AWE enable (32-bit systems)	1 (Only for 32-bits Windows)	Recommended
Blocked process threshold	Default (0)	Optional (Tracing purpose)
Lightweight pooling	Default (0)	Optional (Special purpose)
Max degree of parallelism	1	Recommended
Min / Max Server memory	Fixed value	Recommended
Max worker threads	Lowest possible value	Recommended
Priority boost	Default (0)	Recommended
Recovery interval	Default (0)	Optional (Special purpose)
“special” settings	**recommended value**	**importance**
Lock Pages in Memory	According value	Recommended
Large Pages	Trace flag 834	Optional (High-end only)
Plan cache size	Trace flag 8032	Optional (High-end only)

Caution: Setting one or more of these parameters in the database can lead to issues in other products or Infor LN sessions. Therefore, perform a validation test before you implement.

Affinity (64) I/O Mask

The **affinity I/O mask** option binds SQL Server disk I/O to a specified subset of CPUs. In high-end SQL Server online transactional processing (OLTP) environments, this extension can enhance the performance of SQL Server threads issuing I/Os by placing them on CPUs that does the I/O as well, at windows level. Use it in conjunction with the “affinity mask”. This enhancement does not support hardware affinity for individual disks or disk controllers.

Affinity (64) mask

By segregating SQL Server threads to run on particular processors, Windows 2003 or Windows Server 2008 can better evaluate the system's handling of processes specific to Windows. For example, on an 8-CPU server running two instances of SQL Server (instance A and B), the system administrator could use the **affinity mask** option to assign the first set of 4 CPUs to instance A, and the second set of 4 to instance B. The same applies to segregate SQL Server from Infor LN in a 2-Tier installation.

AWE enable

In SQL Server, you can use the Address Windowing Extensions (AWE) API to provide access to physical memory in excess of the limits set on configured virtual memory. This allows SQL Server to use more than 4 GB of memory on 32-bits version of Windows.

Blocked process threshold

Use the **blocked process threshold** option to specify the threshold, in seconds, at which blocked process reports are generated. The threshold can be set from 0 to 86,400. By default, no blocked process reports are produced. This event is not generated for system tasks or tasks waiting on resources that do not generate detectable deadlocks.

You can define an alert to be implemented when this event is generated. For example, you can (choose to) page the administrator to take appropriate action to handle the blocking situation.

Lightweight pooling

Use the **lightweight pooling** option to reduce the system overhead associated with the excessive context switching, sometimes seen in symmetric multiprocessing (SMP) environments. In case of excessive context switching, lightweight pooling can provide better throughput by performing the context switching inline, which helps reduce user/kernel ring transitions.

Usage of **lightweight pooling** has a small performance improvement for SQL Server with Infor LN on Windows 2000 servers. For Windows 2003/2008 servers, using **lightweight pooling** is not recommended.

Max degree of parallelism

When SQL Server runs on a computer with more than one CPU, it detects the degree of parallelism, that is, the number of processors employed to run a single statement, for each parallel plan execution. You can use the **max degree of parallelism** option to limit the number of processors to use in parallel plan execution. The default value of 0 uses all available processors. Set **max degree of parallelism** to 1 to suppress parallel plan generation. Set the value to a number greater than 1 (up to a maximum of 64) to restrict the maximum number of processors used by a single query execution. Even if a value greater than the number of available processors is specified, the actual number of available processors is used. If the computer has only one processor, the **max degree of parallelism** value is ignored.

When SQL Server is purely used for Infor LN, it is recommended to set max **degree of parallelism** to 1. Most Infor LN queries are short and will not benefit from parallelism. When Infor LN is used with different applications, such as reporting; it is recommended to set the **max degree of parallelism** to a value other than 0. The default value of 0 can heavily affect the Infor LN Performance if a heavy query or a query with performance issues hits the SQL Server.

Min / Max Server memory

Use the two server memory options, **min server memory** and **max server memory**, to reconfigure the memory (in megabytes) in the buffer pool used by an instance of SQL Server. It is recommended to fix the memory of SQL Server by setting **max server memory** and **min server memory** to the same value. SQL Server in dynamic (the default) memory mode has its disadvantages because it keeps competing for memory, if another application exists and will release memory only under heavy memory pressure.

Note: when sizing memory in SQL Server 2012, please be aware that the behavior of “max server memory” is different than previous versions. In previous versions, this configuration option sized only the buffer pool. In SQL Server 2012, it now accounts for many more types of memory allocations within SQL Server. For more information, see http://support.microsoft.com/kb/2663912.

Max worker threads

Use **the max worker threads** option to configure the number of worker threads available to SQL Server processes. SQL Server uses the native thread services of the Windows 2003 and Windows Server 2008 operating systems, so that one or more threads support each network that SQL Server supports simultaneously. Another thread handles database checkpoints, and a pool of threads handles all users.

For this SQL Server parameter, less is more. Set it to the lowest possible value, and note that 128 is the minimum value for SQL Server 2005 and 256 is the minimum value for SQL Server 2008. Setting **the max worker threads** parameter to a high value will result in excessive C/S and decreased performance since most of the Infor LN database requests are short.

Priority boost

Use the **priority boost** option to specify whether SQL Server must run at a higher Windows 2003 or Windows 2008 scheduling priority than other processes on the same computer. If you set this option to 1, SQL Server runs at a priority base of 13 in the Windows 2003 or Windows Server 2008 scheduler. The default is 0, for which the priority base is 7. In special cases and with a highly loaded 2-Tier system, this parameter can be used.

Recovery internal

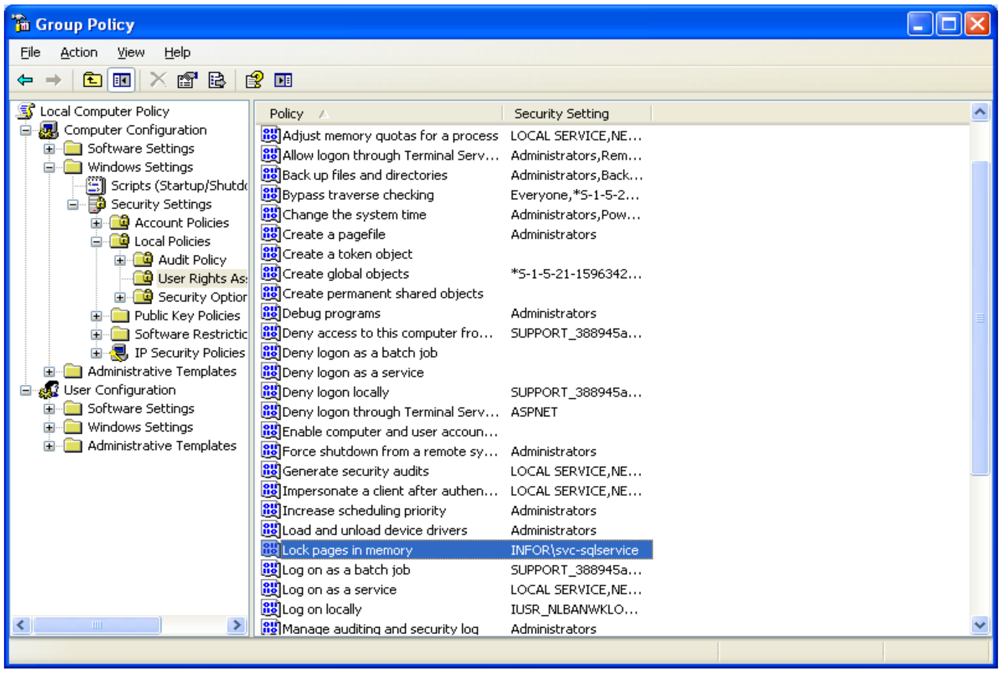
Use the **recovery interval** option to set the maximum number of minutes per database that SQL Server needs to recover databases. Each time an instance of SQL Server starts, it recovers each database, rolls back transactions that did not commit and rolls forward transactions that are committed, but whose changes were not yet written to the disk when the SQL Server stops. This configuration option sets an upper limit on the time it should take to recover each database. The default is 0, indicating automatic configuration by SQL Server. In practice, this means a recovery time of less than one minute and a checkpoint approximately every one minute for active databases.

Keep **recovery interval** set at 0 (self-configuring) unless you notice that checkpoints hinder performance because they occur too frequently. If this is the case, try increasing the value in small increments. Increasing the recovery interval will result in “heavier” checkpoints and more pressure on the I/O subsystem.

Lock Pages in memory

If AWE is enabled on a 32-bit windows system the “Lock Pages in memory” privilege must be enabled. This privilege is important on 64-bit windows systems as well. Memory pages are locked in memory to enable faster allocations. If SQL Server memory pages are “locked” then it is non-pageable, and Windows cannot take it when aggressively trimming the memory. To assign the “Lock pages in memory" privilege you need to modify the Group Policy on the server with Group policy editor (start with gpedit.msc).

Navigate to the User-Rights Assignment folder and add the (service) account to the 'Lock pages in memory" privilege.



Policy	Security Setting
Adjust memory quotas for a process	LOCAL SERVICE,NE...
Allow logon through Terminal Serv...	Administrators,Rem...
Back up files and directories	Administrators,Back...
Bypass traverse checking	Everyone,*S-1-5-2...
Change the system time	Administrators,Pow...
Create a pagefile	Administrators
Create a token object	
Create global objects	*S-1-5-21-1596342...
Create permanent shared objects	
Debug programs	Administrators
Deny access to this computer fro...	SUPPORT_388945a...
Deny logon as a batch job	
Deny logon as a service	
Deny logon locally	SUPPORT_388945a...
Deny logon through Terminal Serv...	ASPNET
Enable computer and user accoun...	
Force shutdown from a remote sy...	Administrators
Generate security audits	LOCAL SERVICE,NE...
Impersonate a client after authen...	LOCAL SERVICE,NE...
Increase scheduling priority	Administrators
Load and unload device drivers	Administrators
Lock pages in memory	INFOR\svc-sqlservice
Log on as a batch job	SUPPORT_388945a...
Log on as a service	LOCAL SERVICE,NE...
Log on locally	IUSR_NLBANWKLO...
Manage auditing and security log	Administrators

Large Pages

What are "Large Pages"? Basically, the idea is to use a larger page size for memory as organized by the kernel. This makes the process of virtual address translation faster. For example, the normal page size for Windows memory is 4Kb on x64 systems. But with large pages, the size is 2Mb.

SQL Server supports the concept of Large Pages when allocating memory for some internal structures and the buffer pool. But the use of large pages for the buffer pool is not for everyone and its usage must be carefully tested and planned. Read the complete section to make the right consideration and decision.

Large Pages are defined at two places within SQL Server:

- **LargePagesAllocator** – SQL Server decides whether it can use the Large Page Support from VirtualAlloc.
- **Large Pages Used by Buffer Pool** – Buffer Pool decides to use the LargePageAllocator to allocate the buffer pool memory.

LargePagesAllocator

When SQL Server is started for the first time, it makes a decision on using large page support from Windows. This decision is based on the following three conditions, which must all be true:

- SQL Server Enterprise Edition
- The computer must have 8 GB or more of physical RAM
- The “Lock Pages in Memory” privilege is set for the service account

If these conditions are true, SQL Server will “initialize” the LargePageAllocator for each memory node on the computer. The ERRORLOG will display the following messages:

```
2009-06-04 12:21:08.16 Server      Large Page Extensions enabled.
2009-06-04 12:21:08.16 Server      Large Page Granularity: 2097152
2009-06-04 12:21:08.21 Server      Large Page Allocated: 32MB
```

The Large Page Granularity is the minimum size of a “large page” on the given Windows Platform. The next message indicates what the LargePageAllocator for SQL Server does when it is initialized. It allocates 32Mb of large page memory to prime the system for any component that needs large pages memory. You will see one of these messages for each memory node (NUMA) created by SQL Server.

Large Pages

Large Pages will be enabled by trace flag 834. When enabled for 64-bit systems, the SQL Server Engine will use the LargePageAllocator to allocate SQL Server memory. From SQL Server 2012 onwards, large pages will be allocated for all SQL Server memory. In previous versions, SQL Server allocated large pages only for the buffer pool.

If the engine detects trace flag 834 enabled at startup, it will allocate memory using the LargePageAllocator. The server will allocate all memory at startup. This is because the allocation of large page with VirtualAlloc() can be very slow. If the buffer pool grows dynamically, it will impact the performance of the standard queries. The memory allocation is a one-time activity.

If these conditions are true, SQL Server will “initialize” the Large Pages for each memory node on the computer. The ERRORLOG will display the following messages (SQL Server 2008 R2):

```
2009-06-04 14:20:31.13 Server      Large Page Extensions enabled.
2009-06-04 14:20:31.13 Server      Large Page Granularity: 2097152
2009-06-04 14:20:31.14 Server      Large Page Allocated: 32MB
2009-06-04 14:20:40.03 Server      Using large pages for buffer pool.
2009-06-04 14:27:56.98 Server      2048 MB of large page memory allocated.
```

SQL Server attempts to allocate the size equal to the minimum of ‘max server memory’ and total physical memory on the computer. If ‘max server memory’ is set to 0, SQL Server will try to allocate the total RAM. For this reason, a suitable value for ‘max server memory’ must be defined. It is advisable to not use total physical RAM to allow the operating system and system cache to function.

Use trace flag 834 together with Infor LN, if the following conditions are true:

- For Windows 2008 R2, or higher.
- When using SQL Server on a dedicated database server or settings ensure that sufficient memory is available for the SQL Server

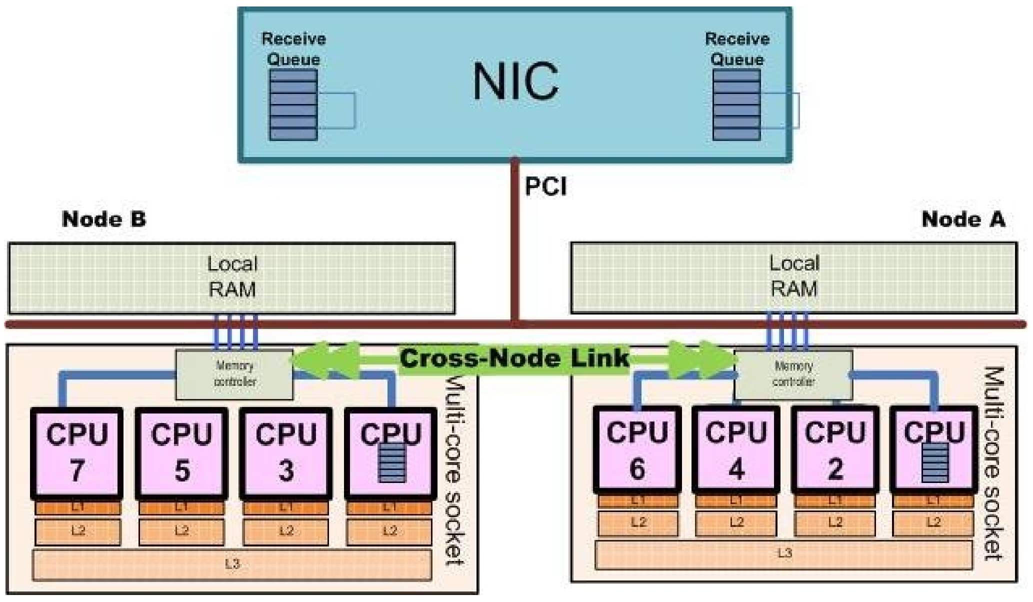
- When using a heavily loaded Infor LN SQL Server database, instance with memory usage above 16 Gb – 32 Gb
- When using a NUMA based system with more than 2 CPU sockets.

It is difficult to provide accurate figures and estimates for the increased performance levels when trace flag 834 is used. On an average, a performance increase in the range of 10 – 50 percent can be expected for heavily loaded SQL Server instances with Infor LN.

SQL Server and NUMA

SQL Server has been designed to take advantage of NUMA-based computers without requiring any application changes. In the past NUMA was used by hardware vendors for “big iron” to resolve SMP-based (Symmetric Multi-Processor) scaling issues. Nowadays, all current AMD and Intel (Nehalem 5500 Series and above) CPUs come with interconnect technology and makes all systems NUMA based computers.

The NUMA architecture is shown in more detail in this figure.



In principle a NUMA based system is broken up in nodes, with in most cases a NUMA-node for each Multi-core socket. This means that the system memory is divided over the nodes as well. Local memory can be accessed fast but memory on other node(s) has to be accessed by the cross-node link and has more latency with a bigger chance of contention.

A lot of tuning recommendations for SQL Server and NUMA can be provided, especially for the high-end usage. Following are the basic recommendations:

- Do not tune anything if the customer requirement is small and you don’t want to deal with additional complexity. SQL Server will function appropriately and it is just a matter of optimization.

- For Infor LN running in a 2-Tier configuration, assign SQL Server to a dedicated NUMA node via “Affinity mask” in sp_configure. SQL Server memory access on the other node(s) is always more expensive, and if CPU capacity of all NUMA-nodes is not required, prevent expensive cross-node access.
- SQL Server memory (set by max memory) is divided over the available NUMA nodes. On most systems currently available, there is no (partial) sharing between memory of the NUMA nodes. This means additional total memory is required for SQL Server to meet the per NUMA node memory requirements, when compared with a non-NUMA system.

Plan cache size

Infor LN benchmarks on SQL Server 2012 showed that SQL server continually compiled query plans again. This should only happen some period after a cold start of the database. Compiling SQL plans is an expensive CPU operation. In the benchmark, we saw this happen when running more than approximately 500 users. If you encounter this issue on you SQL Server 2012 database, it is possible to implement SQL Server trace flag 8032, which uses the SQL Server 2005 RTM cache behavior and in general allows caches to be larger. For more information, see http://msdn.microsoft.com/en-us/library/ms188396.aspx.

Chapter 2 Tuning Infor LN for SQL Server

2

This chapter describes the most important resource settings and performance parameters in Infor LN running on a SQL Server database.

SQL Server db_resource parameters

From porting set 8.6a onwards, the default db_resource values are optimal for most customers. However, the following parameters are important for performance and in certain circumstances it can changed. For 2-tier OLTP and batches (both 2-tier and 3-tier), some db_resource parameters will increase the performance. For more Information, refer to the *Infor Enterprise Server - Technical Reference Guide for Microsoft SQL Server Database Driver (U8173 US)*

The following parameters must be added in the %BSE%/lib/defaults/db_resource file when SQL Server is used as a database for OLTP users:

```
msql_opt_rows:5
msql_serverhost:<dbserver>\erpln
msql_array_fetch:1
msql_array_insert:1
msql_max_arrsz:5
msql_array_insert:1
```

For batches, 2-tier and 3-tier, a separate db_resource file can be used to increase the performance of batches, without affecting the OLTP performance. This can be done via the environment variable USR_DBS_RES to be set only for the user who executes the batches.

Example:

```
-set USR_DBS_RES=lib/defaults/db_resource.batch
```

The following parameters must be set for batches:

```
msql_opt_rows:50
msql_serverhost:<dbserver>\erpln
msql_array_fetch:1
msql_array_insert:1
msql_max_arrsz:50
msql_retained_cursors:200
bdb_max_session_schedule:254
```

msql_opt_rows

The resource msql_opt_rows allows you to specify for the SQL Server, a hint that the first **n** rows must be retrieved fast. This will allow the SQL Server to optimize the fetch request. Based on this value, SQL Server determines a suitable communication buffer size to improve performance. This variable can have a value of **n**, the possible values and the corresponding descriptions are listed below:

- n < 0 No optimization hint
- 0 FASTFIRSTROW table hint
- n > 0 OPTION (FAST n) query hint

From porting set 8.7a.01 onwards, the default value for this option is 1.

msql_retained_cursors

The resource msql_retained_cursors sets the number of inactive (breaked) cursors that must be retained in the list for reuse. These cursors can be reused and save the prepare/bind overhead. It will require additional resources, such as, memory until the cursors are closed and released.

Default value for this option is 20, which is a good starting point for most customers in 2-tier OLTP. For batches it is recommended to increase this parameter to 200.

msql_serverhost

The resource msql_serverhost allows specification of a host name for the driver to locate the SQL Server instance to be used. It is possible to specify network protocol and SQL Server Instance, as displayed below:

```
msql_serverhost:np:<dbserver>\erpln
```

In this example “np:” specifies the “named pipes” network protocol and “dbserver\erpln” specifies the use of “erpln” SQL Server Instance on the server “dbserver”. Refer to Chapter 1 for optimal settings of SQL Server network along with Infor LN.

msql_array_fetch

The resource msql_array_fetch is used to enable or disable the array fetch interface. The valid values are 0 and 1. When set to 0, the array fetch interface is disabled; when set to 1, it is enabled. In porting sets before 8.4a and 6.1c.07.15, this variable must be made available in the environment, preferably in the database definition. This can be confusing and hence can be used as db_resource variable. The default value for this option is 0.

msql_array_insert

The resource msql_array_insert is used to enable or disable the array insert interface; the valid values are 0 and 1. When set to 0, the array insert interface is disabled; when set to 1, it is enabled.

Note: The database driver will disable this option under certain circumstances. For example, if references must be checked or updated, or the application requires immediate response from the driver, no array insert can be executed. The default value for this option is 0.

msql_max_arrsz

If msql_array_insert is enabled, the msql_max_arrsz parameter defines the maximum number of rows inserted into the RDBMS. If msql_array_fetch is enabled, the msql_max_arrsz parameter defines the maximum number of rows fetched from the RDBMS. The default value for this option is 1.

msql_lock_timeout

The resource msql_lock_timeout determines the timeout value, in seconds, for queries, blocked by locks in the database server. The default value is 10, which means "one waits for 10 seconds for the lock to be released". When set to -1, the driver waits indefinitely for the locks to be released; when set to 0, the driver does not wait. It is advised to use this resource carefully. The default should be optimal for customers. The default value for this option is 10.

Database storage and driver files

Database storage files contain parameters that influence the creation of tables and indexes. Only performance topics are discussed. For more Information about these files, consult the driver manuals.

MSQL storage file

The content for creating data and indexes for MSQL storage driver is stored in the msql_storage_param. This file is located in %BSE%\lib\msql. The following file shows the standard storage parameter files of SQL Server:

```
*:*:T:group:0400:FILEGROUP PRIMARY
*:*:I:FILEGROUP PRIMARY
```

FILEGROUP <file group name>

Includes the “on <file group name>” clause in the create table or create index statement during object creation. The following example shows the correct setup of the %BSE%\lib\msql\msql_storage_param to separate data and indexes:

```
*:*:T:group:0400:FILEGROUP PRIMARY
*:*:I1:FILEGROUP PRIMARY
*:*:I:FILEGROUP SECONDARY
```

FILL FACTOR <n>

Includes the “fill factor <n>” clause in the create index statement during index creation. Default fill factor of SQL Server is 0, which indicates a dynamic approach. Although a low fill factor value other than 0 can reduce the requirement to split pages as the index grows, the index will require more storage space and can decrease read performance. Even for a table oriented for many insert and update operations, the number of database reads typically outnumber database writes by a factor of 5 to 10. Therefore, specifying a fill factor other than the default can decrease database read performance by an amount inversely proportional to the fill factor setting. For example, a fill factor value of 50 can cause database read performance to decrease by two times. Read performance is decreased because the index contains more pages, thereby increasing the disk I/O operations required to retrieve the data.

These storage parameters can be used for the following situations:

- “hot” tables with many inserts and deletes
- Lock contention on a specific table

MSQL driver parameters file

The following file shows the standard driver parameter files of SQL Server:

%BSE%\lib\msql\msql_driver_param

```
*:*:T:group:00400::
*:*:I::::
```

NOIDXHINT

NOIDXHINT prevents the driver from generating index hints for the specified table(s); this is strongly recommended for Infor Baan IV. The following example shows the setup of this:

```
*:*:T:group:00400:NOIDXHINT
*:*:I::::
```

For Infor LN it is not recommended to set this value.

Chapter 3 Tracing SQL Server

3

Besides the Infor LN trace possibilities, SQL Server in combination with Windows also has powerful tools to trace an application. Only a small and brief summary is provided. It is difficult to recommend the tool to be used in a specific situation. It is recommended that you always start with a global tool (as Windows Task Manager) and start detailed tracing based on symptoms discovered with global tools.

Windows Task Manager

Never forget the good “old” Windows Task Manager as a starting point for any tracing and tuning exercise. Do not use other “advanced” tools as the root cause in all probability will be visible in the Task manager.

SQL Server Management Studio Activity monitor

The Activity Monitor available in SQL Server Management Studio can be used by database developers and administrators for a quick overview of SQL Server system performance.

Tracing with SQL Server Management Studio Reports

SQL Server Management Studio generates built-in performance reports at both server level and database level. Most reports are based on dynamic management views (DMVs).

Tracing with SQL Server Dynamic Management Views (DMV)

DMVs expose internal memory structures within the SQL Server address space. DMVs replace the virtual tables in SQL Server 2000 (ex sysprocesses). DMVs and dynamic management view functions (DMFs) reflect the server processes itself or all the sessions on the server. DMVs can be used for diagnostics, memory and process tuning, and monitoring the sessions on the server. DMVs alleviate the need to monitor the server with tools, such as Profiler and Performance Monitor (Perfmon), to diagnose performance issues. Additionally, DMVs can provide Information about performance issues even after an issue has occurred.

Tracing with sqlcmd and showplan options

Usually the graphical plan is easy to use but in certain scenarios it is beneficial to use command prompt utilities.

Executing the following qptool query:

```
qptool -q "select item, dsca from tcibd001 where seak='COST SET 1'"
'         COST1                                 ' 'Cost Set 1

           '
```

The SQL query can be executed on the command prompt with the “sqlcmd” utility. Sqlcmd -? will show all the options available to sqlcmd.

When SET_SHOWPLAN_TEXT {ON|OFF} is on, Information is displayed about the query execution plan used. The query is not executed:

```
sqlcmd -S <dbserver>\erpln -d erplndb -U sa -P <password>
1> SET SHOWPLAN_TEXT ON
2> GO
1> select t_item, t_dsca from dbo.ttcibd001090 where t_seak="COST SET 1"
2> GO
StmtText
-------------------------------------------------------------------------
 select t_item, t_dsca from dbo.ttcibd001090 where t_seak="COST SET 1"

(1 rows affected)
StmtText
---------------------------------------------------------------------------
 |--Nested Loops(Inner Join, OUTER REFERENCES:([baandb].[dbo].[ttcibd001090
   |--Index Seek(OBJECT:([baandb].[dbo].[ttcibd001090].[Ittcibd001090_2a]),
   |--Clustered Index Seek(OBJECT:([baandb].[dbo].[ttcibd001090].[Ittcibd00

(3 rows affected)
```

The example does not show the complete result set as it becomes unreadable due to wrapping.

When SET_SHOWPLAN_ALL {ON|OFF} is set, Information is displayed about the query execution plan used. The query is not executed. To prevent wrapping to a certain extent, it is important to set the command prompt window to a high buffer and window scroll bar with the following:

```
sqlcmd -S <dbserver>\erpln -d baandb -U sa -P <password>
1> set SHOWPLAN_ALL ON
2> go
1>  select t_item, t_dsca from dbo.ttcibd001090 where t_seak="COST SET 1"
2> go
StmtText
                                                                  DefinedValues
--------------------------------------------------------------------------------------
-----
----------------------------------------------------------------- --------------------
-----
 select t_item, t_dsca from dbo.ttcibd001090 where t_seak="COST SET 1"
                                                                  NULL
  |--Nested Loops(Inner Join, OUTER
REFERENCES:([baandb].[dbo].[ttcibd001090].[t_item], [Ex
                                                                  NULL
       |--Index Seek(OBJECT:([baandb].[dbo].[ttcibd001090].[Ittcibd001090_2a]),
SEEK:([baan
]=N'COST SET 1') ORDERED FORWARD
[baandb].[dbo].[ttcibd001
       |--Clustered Index
Seek(OBJECT:([baandb].[dbo].[ttcibd001090].[Ittcibd001090_1a]), S
]=[baandb].[dbo].[ttcibd001090].[t_item]) LOOKUP ORDERED FORWARD
[baandb].[dbo].[ttcibd001

(4 rows affected)
```

SET SHOWPLAN_ALL returns Information as a set of rows that form a hierarchical tree representing the steps taken by the SQL Server query processor, as it executes each statement. Each statement reflected in the output contains a single row with the text of the statement, followed by several rows with the details of the execution steps. The output columns are displayed in this table:

Column name	Description
StmtText	For rows not of type PLAN_ROW, this column contains the text of the Transact-SQL statement. For rows of type PLAN_ROW, this column contains a description of the operation. This column contains the physical operator and can also contain the logical operator. This column can also be followed by a description that is determined by the physical operator. For more Information, see Logical and Physical Operators Reference.
StmtId	Number of the statement in the current batch.
NodeId	ID of the node in the current query.
Parent	Node ID of the parent step.
PhysicalOp	Physical implementation algorithm for the node. For rows of type PLAN_ROWS only.

Column name	Description
LogicalOp	This node represents relational algebraic operator. For rows of type PLAN_ROWS only.
Argument	Provides extra Information about the operation being performed. The contents of this column depend on the physical operator.
DefinedValues	Contains a comma-separated list of values introduced by this operator. These values might be computed expressions which were present in the current query, such as in the SELECT list or WHERE clause, or internal values introduced by the query processor to process this query. These defined values can be referred to, again, within the query. For rows of type PLAN_ROWS only.
EstimateRows	Estimated number of rows of output produced by this operator. For rows of type PLAN_ROWS only.
EstimateIO	Estimated I/O cost for this operator. For rows of type PLAN_ROWS only.
EstimateCPU	Estimated CPU cost for this operator. For rows of type PLAN_ROWS only.
AvgRowSize	Estimated average row size (in bytes) of the row being passed through this operator.
TotalSubtreeCost	Estimated (cumulative) cost of this operation and all child operations.
OutputList	Contains a comma-separated list of columns being projected by the current operation.
Warnings	Contains a comma-separated list of warning messages relating to the current operation. Warning messages might include the string "NO STATS:()" with a list of columns. This warning message implies that the query optimizer attempted to make a decision on the statistics for this column, but none were available. Consequently, the query optimizer had to make a guess, resulting in the selection of an inefficient query plan. For more Information about creating or updating column statistics (which help the query optimizer choose a more efficient query plan). This column might include the string "MISSING JOIN PREDICATE", which means a join (involving tables) is taking place without a join predicate. Accidentally dropping a join predicate might result in a query which takes much longer to run than expected, and returns a huge result set. If this warning is present, verify if the absence of a join predicate is intentional.
Type	Node type. For the parent node of each query, this is the Transact-SQL statement type, such as SELECT, INSERT, EXECUTE, and so on. For subnodes representing execution plans, the type is PLAN_ROW.
Parallel	0 = Operator is not running in parallel. 1 = Operator is running in parallel.

Column name	Description
EstimateExecutions	Estimated number of times this operator will be executed while running the current query.

The SET SHOWPLAN_TEXT and SET SHOWPLAN_ALL statements display Information about the query execution plan adopted by the query optimizer. Neither statement allows the query to be executed. The SET STATISTICS PROFILE {ON|OFF} does not hinder the execution of the query. Apart from returning the same Information as SET SHOWPLAN_ALL, it also displays two additional columns: Rows and executes. “Rows” contains the numbers of rows returned, and “Executed” contains the actual number of times the operator executed the query.

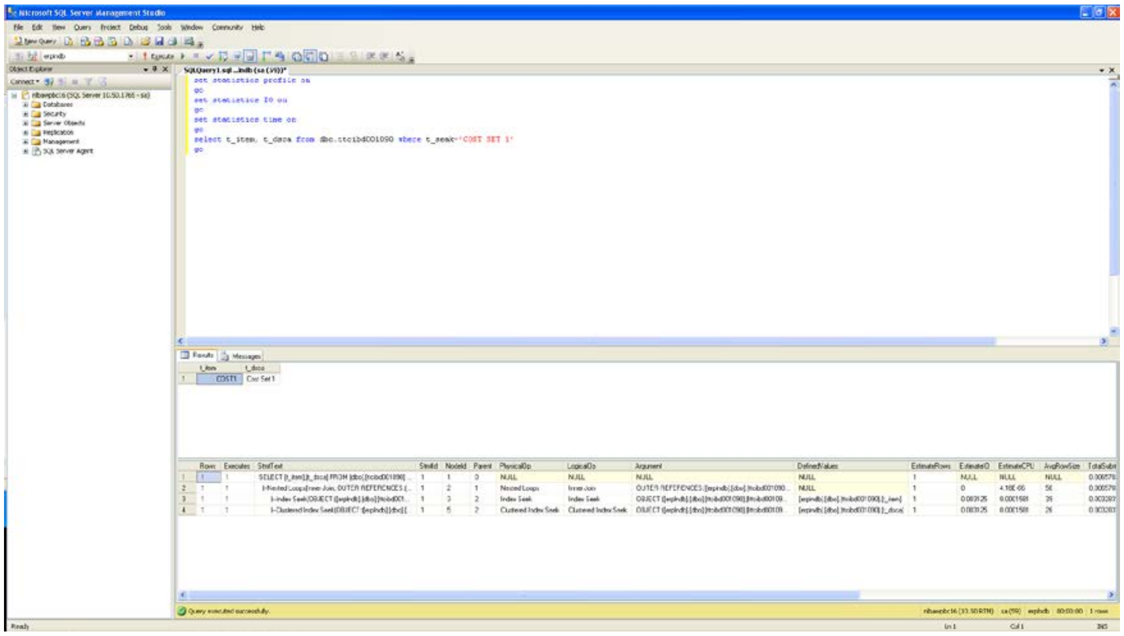
Another useful SET statement when investigating query optimization is SET STATISTICS IO {ON|OFF}. This displays the count of table accesses, logical and physical reads, and read ahead reads for each T-SQL statements.

The SET STATISTICS TIME {ON|OFF} statement displays the time in milliseconds to parse the statement, compile the query, and execute the query.

Tracing with SQL Server management studio

One of the best methods to trace the SQL Server is to use the SQL Server Management Studio.

A useful feature of the SQL Server management studio is that you can run a query in “sqlcmd mode” and display the same output as the “sqlcmd utility”. This ensures that the Information is much better formatted, as shown in this screenshot:


Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Community Help
New Query
Execute
Object Explorer
SQLQuery1.sql ...lndb (sa (59))*
Results
Messages

```
set statistics profile on
go
set statistics IO on
go
set statistics time on
go
select t_item, t_dsca from dbo.ttcibd001090 where t_seak='COST SET 1'
go
```

	t_item	t_dsca
1	COST1	Cost Set 1

	Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubt
1	1	1	SELECT [t_item],[t_dsca] FROM [dbo].[ttcibd001090] ...	1	1	0	NULL	NULL	NULL	NULL	1	NULL	NULL	NULL	0.006578
2	1	1	|--Nested Loops(Inner Join, OUTER REFERENCES:(...	1	2	1	Nested Loops	Inner Join	OUTER REFERENCES:([erplndb].[dbo].[ttcibd001090]...	NULL	1	0	4.18E-06	56	0.006578
3	1	1	|--Index Seek(OBJECT:([erplndb].[dbo].[ttcibd001...	1	3	2	Index Seek	Index Seek	OBJECT:([erplndb].[dbo].[ttcibd001090].[ttcibd00109...	[erplndb].[dbo].[ttcibd001090].[t_item]	1	0.003125	0.0001581	39	0.003283
4	1	1	|--Clustered Index Seek(OBJECT:([erplndb].[dbo].[...	1	5	2	Clustered Index Seek	Clustered Index Seek	OBJECT:([erplndb].[dbo].[ttcibd001090].[ttcibd00109...	[erplndb].[dbo].[ttcibd001090].[t_dsca]	1	0.003125	0.0001581	26	0.003283

Query executed successfully. nlbaerpbc16 (10.50 RTM) sa (59) erplndb 00:00:00 1 rows

Ready Ln 1 Col 1 INS

The tab messages display the additional statistics Information, detailed below:

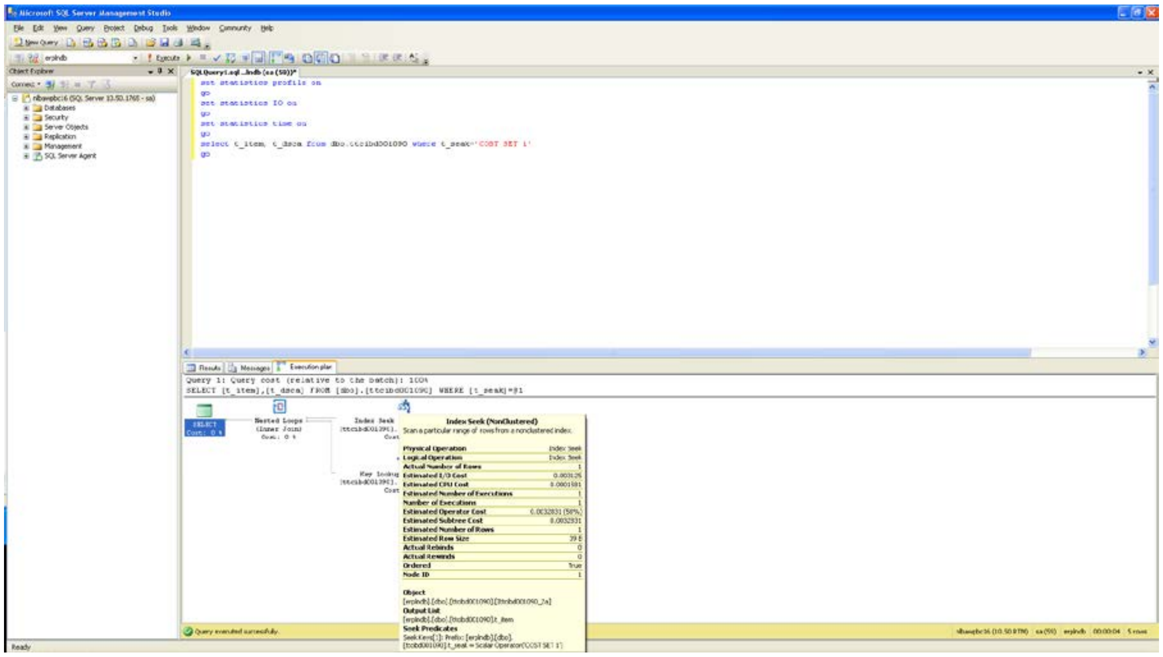
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.

(1 row(s) affected)
Table 'ttcibd001090'. Scan count 1, logical reads 7, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

(4 row(s) affected)

SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
```

Switching the “Include Actual Execution Plan”, under the “query” tab, displays the graphical query execution plan. An example of trace output in SQL Server Management Studio is shown in this screenshot:


Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Community Help
New Query
Execute
Object Explorer
SQLQuery1.sql ...lndb (sa (59))*
Results
Messages
Execution plan
SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
Index Seek
Key lookup
Query executed successfully.
Ready

```
set statistics profile on
go
set statistics IO on
go
set statistics time on
go
select t_item, t_dsca from dbo.ttcibd001090 where t_seak='COST SET 1'
go
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT [t_item],[t_dsca] FROM [dbo].[ttcibd001090] WHERE [t_seak]=@1
```

Index Seek (NonClustered)
Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Number of Rows	1
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Number of Executions	1
Number of Executions	1
Estimated Operator Cost	0.0032831 (50%)
Estimated Subtree Cost	0.0032831
Estimated Number of Rows	1
Estimated Row Size	39 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	1

Object
[erplndb].[dbo].[ttcibd001090].[ttcibd001090_2a]
Output List
[erplndb].[dbo].[ttcibd001090].t_item
Seek Predicates
Seek Keys[1]: Prefix: [erplndb].[dbo].[ttcibd001090].t_seak = Scalar Operator('COST SET 1')

Tracing with SQL Server Profiler

The SQL Server Profiler is a powerful SQL Server tracing tool. Creating a SQL Server Profiler snapshot of a badly performing query or multiple statements will help you identify the reason for the performance issue.

A new powerful feature of the SQL Server Profiler for SQL Server 2005 and onwards is the option to correlate a Profiler trace with Performance monitor (perfmon) data. Performance impact on the database server correlates directly to the Profiler trace and vice versa.

Tracing with SQLDiag utility

SQLDiag is a diagnostics collection utility that can run either from the command prompt or as a service. SQLDiag can gather the following Information:

- Windows Performance logs
- Windows Event logs
- SQL Server Profiler traces
- SQL Server blocking Information
- SQL Server Configuration Information

It is recommended that you gather performance Information first (e.g. a customer side Information) to analyze it later. This will ensure that all the required Information is easily available. You can also use the PAL tool utility to analyze the Information.

Tracing with PAL Tool

When you have a performance issue and are not sure which performance counters to gather or how to analyze them, you must use the PAL tool (Performance Analysis of Logs). It is a powerful tool that reads into a performance monitor counter log and analyzes it using known thresholds.

Features

- Threshold files for most of the major Microsoft products such as IIS, SQL Server, etc.
- An easy to use GUI interface which enables creating batch files for the PAL.ps1 script.
- A GUI editor for creating or editing your own threshold files.
- Creates an HTML based report which is easy to copy/paste into other applications.
- Analyzes performance counter logs for thresholds using thresholds that change their criteria based on the computer's role or hardware specs.

Tracing with Performance monitor

To monitor the utilization of system resources, use Performance Monitor (perfmon). Collect and view real-time performance data in the form of counters, for server resources such as processor and memory use, and for many SQL Server resources such as locks and transactions.

The most useful SQL Server resources in the Performance monitor are:

- SQLServer: Access Methods.
- SQLServer: Buffer Manager.
- SQLServer: Cursor Manager by Type.
- SQLServer: Cursor Manager Total.
- SQLServer: Databases.
- SQLServer: General Statistics.
- SQLServer: Latches.
- SQLServer: Locks.
- SQLServer: Memory Manager.
- SQLServer: SQL Statistics.
- SQLServer: Wait Statistics.

Tracing with Server-Side Traces

A different name for Server-Side tracing is tracing using stored procedures; that is what actually exactly happens with Server-Side traces. The SQL Profiler is a powerful tool, but the graphical UI that displays all the captured events utilizes most of the CPU resources. For long running traces on loaded SQL Servers systems, Server-Side traces are a good solution.

The stored procedures used for these tasks are documented. However, the easiest way to create scripts that utilize them is to create a trace using the SQL Profiler graphical UI, and then from the File menu, select Export.

Following are the system stored procedures to be used when creating and managing a trace:

- sp_trace_create
- sp_trace_generateevent
- sp_trace_setevent
- sp_trace_setfilter
- sp_trace_setstatus

Tracing with SQL Nexus

SQL Nexus is a tool that helps you identify the root cause of SQL Server performance issues. It loads and analyzes performance data collected by SQLDiag. It can significantly reduce the amount of time you spend to analyze the data manually.

Before working with the SQL Nexus tool, it is recommended that you gain knowledge of other performance topics and tools of SQL Server.

SQL Nexus features:

- Fast, easy data loading: You can quickly and easily load SQL Trace files; T-SQL script output, including SQL DMV queries; and Performance Monitor logs into a SQL Server database for analysis. All three facilities use bulk load APIs to insert data quickly. You can also create your own importer for a custom file type.
- View loaded data via reports: Once the data is loaded, you can generate several different charts and reports to analyze it.
- Trace aggregation to display the TOP N most expensive queries (using ReadTrace)
- Wait stats analysis to view blocking and other resource contention issues (based on SQL Perf Stats)
- Full-featured reporting engine: SQL Nexus uses the SQL Server Reporting Services client-side report viewer (it does not require an RS instance). You can create reports for Nexus from either the RS report designer or the Visual Studio report designer. You can also modify the reports that ship with Nexus using either facility. Zoom in/Zoom out to view server performance during a particular time window. Expand/collapse report regions (subreports) for easier navigation of complex data. Export or email reports directly from SQL Nexus. Nexus supports exporting to Excel, PDF, and several other formats.
- Extensibility: You can use the existing importers to load the output from any DMV query into a table, and any RS reports you drop in the Reports folder will automatically show up in the reports task pane. If you want, you can even add a new data importer for a new data type. SQL Nexus will automatically “fix up” the database references in your reports to reference the current server and database, and it will provide generic parameter prompting for any parameters your reports support.