

# Did You Know? It's Easy to Simplify SQL Server Management.

---

Helpful SQL Server Tips: A Quest E-book

# Contents

- Query Tuning Strategies for Microsoft® SQL Server** page 3  
**By Kevin Kline**
- Knowing as many tuning tricks and techniques as possible expands your options when tuning for performance. Read this chapter to discover several basic elements you can use to successfully tune queries. In addition, you'll be exposed to scenarios where poor performance is common, then learn our recommendations for improvement.
- Best Practices in Index Maintenance – Fighting the Silent Performance Killers** page 35  
**By Brent Ozar**
- Index fragmentation is a serious issue in every data center because it can significantly degrade performance. In this chapter, you'll get an introduction to the problem, find out why native and custom-coded approaches are ineffective, and learn how Quest Capacity Manager for SQL Server delivers a robust and reliable solution.
- Ten Things DBAs Need to Know About Storage** page 46  
**By Brent Ozar**
- Sometimes SQL Server database administrators (DBAs) and storage area network (SAN) administrators are at odds when discussing storage and capacity. How can they meet half way? The key is communication. This chapter reveals the top 10 things DBAs should know about storage to help them work effectively with their SAN colleagues.
- Top Ten Things You Should Know About Optimizing SQL Server Performance** page 65  
**By Patrick O'Keeffe**
- Performance optimization on SQL Server is difficult. While a vast array of information exists about how to address performance problems in general, there is not much specific information available. In this chapter, learn 10 details you need to know about SQL Server performance. Each detail is a nugget of practical knowledge that can be immediately applied to your environment.
- Quest Performance Management Solutions for SQL Server** page 94  
**By Quest Software**
- This chapter will show you the value of using Quest's SQL Server performance management products, including Foglight® Performance Analysis for SQL Server and Spotlight® on SQL Server Enterprise. Learn how you can use this solution set to resolve CPU bottlenecks, I/O bottlenecks, and memory pressure issues, as well as simplify TempDB troubleshooting.

# Query Tuning Strategies for Microsoft<sup>®</sup> SQL Server<sup>®</sup>

---

Written by  
Kevin Kline, Microsoft MVP since 2004  
Technical Strategy Manager, Quest Software

© 2009 Quest Software, Inc.  
**ALL RIGHTS RESERVED.**

This document contains proprietary information protected by copyright. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose without the written permission of Quest Software, Inc. (“Quest”).

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST’S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters  
LEGAL Dept  
5 Polaris Way  
Aliso Viejo, CA 92656  
**www.quest.com**  
email: **legal@quest.com**

Refer to our Web site for regional and international office information.

## Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, ChangeManager, Defender, DeployDirector, Desktop Authority, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, IWatch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, MultSess, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point,Click,Done!, PowerGUI, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, Security Lifecycle Map, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer, vRanger, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Updated—September 2009

# Contents

---

- Overview .....6
- What's My Query Doing? And Why Is It Taking So Long? .....7
  - SET STATISTICS I/O .....8
  - SET STATISTICS TIME .....9
    - What's a Test Jig? I Don't Like Dancing! .....11
  - SET SHOWPLAN .....13
    - An Execution Plan Does Not Require an Executioner .....13
    - Yes, Sir! SARG, Sir! .....14
    - Which Is Better? Comparing Two Variants as Illustrated by SEEK or SCAN Operations .....14
- Special Case Scenarios for Query Tuning .....17
  - Functions and Expressions That Suppress Indexes .....17
  - Head Fakes to the Query Optimizer .....18
  - Subqueries Optimization .....19
  - UNION vs. UNION ALL .....21
  - UPDATE...FROM and DELETE...FROM .....22
  - TOP .....24
  - Let's All JOIN Hands and Sing: Understanding the Impact of Joins .....27
  - SET NOCOUNT ON .....30
  - Querying Against Composite Keys .....31
- Summary .....33
- About the Author .....34

# Overview

---

This white paper offers useful techniques for improving queries in Microsoft SQL Server 2008. There are always a large number of tips and techniques applicable in narrow classes of programming tasks, each one offering a small improvement in performance. Knowing as many of these tuning tricks and techniques as possible expands your options when tuning for performance. In addition, knowing an effective process for analyzing query performance and behavior is an essential skill for any SQL Server professional.

The white paper introduces several basic elements that I've used with some success for tuning queries. In addition, it describes a handful of scenarios where poor performance is common, and provides recommendations for improvement.

The basic elements of query tuning that are covered include:

1. SET commands that show you what a query is doing
2. DBCC commands that help construct a useful "test jig" for query tuning
3. Key elements of an execution plan to consider

Most examples are based on either the venerable PUBS database, the NORTHWIND database, or on standard system tables. I have greatly expanded the size of the tables used in the PUBS database, adding tens of thousands of rows to many tables. You can find the PUBS database at <http://codeplex.com/SqlServerSamples>.

# What's My Query Doing? And Why Is It Taking So Long?

There's a lot of instrumentation in SQL Server 2008 that helps you see what your query is doing behind the scenes to retrieve a given result set. You can use trace files, queries against Dynamic Management Views (DMVs) and Dynamic Management Functions (DMFs), and the many graphic features of SQL Server Management Studio (SSMS) to better illustrate the behaviors of a given SELECT statement.

As a long-time SQL Server tuner, I find that many of the old (dare I say “antiquated”) methods of assessing a query are in fact the easiest and most effective. And by effective, I mean that they return the most actionable information in the least amount of time and with the least amount of personal, human intervention. Yes, the graphic tools can provide more information than the old scripted SET statements available in Transact-SQL. However, the graphic tools require that you put your hand on the mouse and click—a lot. That means the information is neither immediately actionable (because you have to point and click a lot to get the information) nor is it something that you can easily script to run in the off-hours when you're not at your desk.

The commands I like to use are:

## **SET STATISTICS I/O**

Shows the overall I/O of the query, including the number of scans performed, the number of logical reads performed (reads from cache), the number of physical reads performed (reads from disk), and the number of read-aheads performed (the number of pages placed in cache in anticipation of future reads). Since I/O is often one of the biggest bottlenecks for a query, it's important to know its overall I/O utilization and to compare the I/O utilization of two (or more) alternative queries.

**Note:** SET STATISTICS I/O can return inaccurate I/O counts on queries that involve LOBs.

## **SET STATISTICS TIME**

Shows the total elapsed time (i.e. the round-trip time) of the query, as well as the CPU time consumed to parse, compile and execute the query. The round-trip time is dependent upon the total activity on the server, while the CPU time is independent of the total activity on the server. (Note—SET STATISTICS TIME may return inaccurate results for queries on servers running in fibre mode.)

## **SET SHOWPLAN\_ALL**

Shows the estimated (not actual) execution plan chosen for a given query in hierarchical format that is representative of the steps taken by the query engine to process the query. The pipe marks in the output indicate the general level of the statement, with more of the first actions of the query appearing at the bottom of the output and working their way upward. You can use SET SHOWPLAN\_TEXT for a subset of output returned by SET SHOWPLAN\_ALL, which is useful when performing query tuning via a scripted method, such as the OSQL utility. Conversely, you can use the SET SHOWPLAN\_XML statement to get even more data about the query than that provided by SET SHOWPLAN\_ALL. It's up to you as to which you might like to use.

It's important to remember that, as with any SET statement, the statement remains in effect until explicitly disabled with the OFF subclause once it's been enabled with the ON subclause. For example, the following Transact-SQL code will show the I/O, time and execution plan of only the single query:

```
SET STATISTICS IO ON
SET STATISTICS TIME ON
SET SHOWPLAN_ALL ON
GO
```

```

SELECT st.stor_name AS 'Store',
       ISNULL((SELECT SUM(bs.qty)
              FROM sales AS bs
              WHERE bs.stor_id = st.stor_id), 0)
       AS 'Books Sold'
FROM   stores AS st
WHERE  st.stor_id IN
      (SELECT DISTINCT stor_id
       FROM sales)
GO

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
SET SHOWPLAN_ALL OFF
GO

```

In the preceding query, if the final SET...OFF statements were not included, all subsequent statements would also return the I/O, time and execution plans of those statements. Note that the results are displayed in the MESSAGES tab of SSMS and not in the RESULTS tab.

Remember that the SET SHOWPLAN\_ALL ON statement only displays the estimated execution plan. If you wish to see the actual execution plan of a query, use the SET STATISTICS PROFILE statement.

## SET STATISTICS I/O

The command SET STATISTICS IO ON forces SQL Server to report actual I/O activity on executed transactions. Once the option is enabled, every query thereafter produces additional output that contains I/O statistics. To disable the option, execute SET STATISTICS IO OFF.

For example, the following script obtains I/O statistics for a simple query counting rows of the “Employees” table in the NORTHWIND database:

```

SET STATISTICS IO ON
GO
SELECT COUNT(*) FROM employees
GO
SET STATISTICS IO OFF
GO

```

**Results:**

```

-----
2977

```

Table 'Employees'. Scan count 1, logical reads 53, physical reads 0, read-ahead reads 0. The scan count tells us the number of scans performed. Logical reads show the number of pages read from the cache. Physical reads show the number of pages read from the disk. Read-ahead reads indicate the number of pages placed in the cache in anticipation of future reads.

Additionally, you would execute a system stored procedure to obtain table size information for your analysis:

```
sp_spaceused employees
```



## Results:

name	rows	reserved	data	index_size	unused
Employees	2977	2008 KB	1504 KB	448 KB	56 KB

What can you tell by looking at this information?

The query did not have to scan the whole table. The volume of data in the table is more than 1.5 megabytes, yet it took only 53 logical I/O operations to obtain the result. This indicates that the query has found an index that could be used to compute the result, and scanning the index took less I/O than it would take to scan all data pages.

Index pages were mostly found in the data cache since the physical reads value is zero. This is because the query was executed shortly after other queries on the Employees table, and the table and its index were already cached. Your mileage may vary.

SQL Server has reported no read-ahead activity. In this case, data and index pages were already cached. A table scan on a large table read-ahead would probably kick in and cache necessary pages before your query requested them. Read-ahead turns on automatically when SQL Server determines that your transaction is reading database pages sequentially. A separate SQL Server connection runs ahead of your process and caches data pages for it. Configuration and tuning of read-ahead parameters is beyond the scope of this paper.

In this example, the query was executed as efficiently as possible. No further tuning is required.

## SET STATISTICS TIME

Elapsed time of a transaction is a volatile measurement, since it depends on activity of other users on the server. However, it provides some real measurement, compared to the number of data pages, which doesn't mean anything to your users. They are concerned about seconds and minutes they spend waiting for a query to come back, not about data caches and read-ahead efficiency. The SET STATISTICS TIME ON command reports the actual elapsed time and CPU utilization for every query that follows. Executing SET STATISTICS TIME OFF suppresses the option.

```
SET STATISTICS TIME ON
GO
SELECT COUNT(*) FROM titleauthors
GO
SET STATISTICS TIME OFF
GO
```

## Results:

```
SQL Server Execution Times:
    cpu time = 0 ms. elapsed time = 8672 ms.
SQL Server Parse and Compile Time:
    cpu time = 10 ms.
```

```
-----
25

(1 row(s) affected)
```

```
SQL Server Execution Times:
    cpu time = 0 ms. elapsed time = 10 ms.
SQL Server Parse and Compile Time:
    cpu time = 0 ms.
```

The first message reports a somewhat confusing elapsed time value of 8,672 milliseconds. This number is not related to the script and indicates the amount of time that has passed since the previous command execution. You may disregard this first message. It took SQL Server only 10 milliseconds to parse and compile the query. It took 0 milliseconds to execute it (shown after the result of the query). What this really means is that the duration of the query was too short to measure. The last message that reports parse and compile time of 0 ms, refers to the SET STATISTICS TIME OFF command (that's the time it took to compile it). You may disregard this message since the most important messages in the output are highlighted.

Note that elapsed and CPU time are shown in milliseconds. The numbers may vary on your computer between runs of the query because the time values are dependent on total server load. In other words, every time you execute this script you may get slightly different statistics depending on what else your SQL Server was processing at the same time.

If you need to measure elapsed duration of a set of queries or a stored procedure, it may be more practical to implement it programmatically (shown below). The reason is that the STATISTICS TIME reports duration of every single query and you have to add things up manually when you run multiple commands. Imagine the size of the output and the amount of manual work in cases when you time a script that executes a set of queries thousands of times in a loop!

Instead consider the following script to capture time before and after the transaction and report the total duration in seconds (you may use milliseconds if you prefer):

```
DECLARE @start_time DATETIME
SELECT @start_time = GETDATE()

< any query or a script that you want to time, without a GO >

SELECT 'Elapsed Time, sec' = DATEDIFF( second, @start_time, GETDATE() )
GO
```

If your script consists of several steps separated by GO, you can't use a local variable to save the start time. A variable is destroyed at the end of the step, defined by the GO command, where it was created. But you can preserve start time in a temporary table like this:

```
CREATE TABLE #save_time ( start_time DATETIME NOT NULL )
INSERT #save_time VALUES ( GETDATE() )
GO
< any script that you want to time (may include GO) >
GO
SELECT 'Elapsed Time, sec' = DATEDIFF( second, start_time, GETDATE() )
FROM #save_time
DROP TABLE #save_time
GO
```

Remember that SQL Server's DATETIME datatype stores time values in three millisecond increments. It is impossible to get more granular time values than those using the DATETIME datatype. In SQL Server 2008, you can opt for using the DATETIME2 datatype if you need greater granularity with the time measurement.

If you'd like to discover more about how to read and interpret execution plans, here are a few articles and posts:

- [http://sqlserverpedia.com/wiki/Examining\\_Query\\_Execution\\_Plans](http://sqlserverpedia.com/wiki/Examining_Query_Execution_Plans)
- <http://sqlserverpedia.com/blog/sql-server-2005/three-kinds-of-execution-plans/>
- <http://sqlserverpedia.com/blog/sql-server-bloggers/understanding-statistics-io-2/>

## What's a Test Jig? I Don't Like Dancing!

No, a test jig is not a kind of dance. Although the word jig commonly refers to certain Irish and English dances from Jane Austin's day, this usage in software engineering comes from the old industrial days when a jig referred to a special kind of box. The box served as a framework to hold work firmly in place so that an artisan could mill, drill, or otherwise perform precision work with both hands. And so, you're doing something similar with your Transact-SQL code. People also refer to this sort of code as a test harness.

Typically, you'll use a test jig to ensure that all of the types of cache, both buffer cache for data and procedure cache for objects, are uniformly clean. This enables you to ensure that an improvement in query performance is due to the change you made in the code and not due to unexpectedly cached objects or data. There are three commands that can help you control buffer and procedure caching while you test the SQL query:

```
DBCC FREEPROCCACHE [ ( { plan_handle | sql_handle | pool_name } ) ] [ WITH NO_INFOMSGS ]
```

Clears the entire procedure cache of the SQL Server 2008 instance if you provide no parameters. Otherwise, it clears the procedure cache of the single execution plan from the cache (using the supplied plan handle or SQL handle) or all workload groups for the named resource pool on SQL Server 2008 Enterprise Edition using resource governor. On SQL Server 2005, the statement works only to clear the entire cache.

```
DBCC FLUSHPROCINDB(<DBID>)
```

Clears all execution plans from the procedure cache of the SQL Server instance for the specified database ID.

```
DBCC DROPCLEANBUFFERS [ WITH NO_INFOMSGS ]
```

Clears the buffer cache (i.e., data) of all clean buffers. To ensure that the buffer cache contains only clean buffers, first execute the CHECKPOINT statement to force all dirty pages to disk. After a CHECKPOINT, the DROPCLEANBUFFERS statement removes all data from the buffer cache.

The subclause WITH NO\_INFOMSGS simply suppresses informational messages returned by the DBCC commands.

**Important:** Do not run these commands on a production system. Although cleaning the procedure and buffer cache are very important for testing the development of a query, you should not execute these commands hastily since they will literally clear the caches of the current instance of SQL Server. If the system is used in any production capacity, then queries may experience a precipitous decline in performance until the applicable data and/or objects are reloaded into the cache. In general, you will only perform these statements within your local development environment.

So building on the example at the end of the previous section, your test jig should look like this:

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE
GO

SET STATISTICS IO ON
SET STATISTICS TIME ON
GO

SET SHOWPLAN_ALL ON
GO

SELECT st.stor_name AS 'Store',
       ISNULL((SELECT SUM(bs.qty)
              FROM sales AS bs
              WHERE bs.stor_id = st.stor_id), 0)
       AS 'Books Sold'
```

```

FROM stores AS st
WHERE st.stor_id IN
      (SELECT DISTINCT stor_id
       FROM sales)
GO

```

```

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
GO

```

```

SET SHOWPLAN_ALL OFF
GO

```

Notice that the SET SHOWPLAN\_ALL statement is separated by its own GO delimiter. That's because the SET SHOWPLAN\_ALL statement must be executed alone in its own Transact-SQL batch.

There are a number of ways that you can get additional information about what is happening with your queries on an instance of Microsoft SQL Server 2008. While a full discussion of these DMVs and DMFs are beyond the scope of this white paper, you can find the query PLAN\_HANDLE using the following four DMVs: sys.dm\_exec\_cached\_plans, sys.dm\_exec\_requests, sys.dm\_exec\_query\_memory\_grants and sys.dm\_exec\_query\_stats. You can find the query SQL\_HANDLE in these five DMVs: sys.dm\_exec\_query\_stats, sys.dm\_exec\_requests, sys.dm\_exec\_cursors, sys.dm\_exec\_xml\_handles and sys.dm\_exec\_query\_memory\_grants. Resource governor POOL\_NAME values are found in the DMV sys.dm\_resource\_governor\_resource\_pools.

Remember, as with any DMV, the data retrieved by a query against the DMV represents the current conditions of the SQL Server instance. Therefore, a query against a DMV may not return information about a particular query because it has aged out of the cache or the cache has been cleaned.

### **CONTEXT Isn't Just a Prison Story**

There are a lot of good programming practices that are commonly used; for example, including a header comment block at the beginning of your Transact-SQL program stating who wrote the program, when it was written, and why. Later, other programmers will update the header block with their own comments describing changes and updates to the original program.

Here's one tip that you might want to incorporate into your standardized coding practices. One element of good programming that isn't very commonly used is to set the CONTEXT\_INFO value for a query or a Transact-SQL program. Context information is optional trace information which is created for a session using the SET CONTEXT\_INFO statement, thereby associating up to 128 bytes of binary data with the current session or connection. The context information data, which cannot be null, is then stored in the CONTEXT\_INFO columns of sys.dm\_exec\_requests, sys.dm\_exec\_sessions and sys.sysprocesses. (In SQL Server 2000, the same data is stored in master.dbo.sysprocesses.)

```

A basic implementation of CONTEXT_INFO might appear as follows:
DECLARE @Ctxt varbinary(128)
SET @Ctxt = CONVERT(varbinary(128), 'Mary''s session')
SET CONTEXT_INFO @Ctxt
GO

```

You can then see the context information when you check on SQL Server's activity in SSMS under Management >> Activity Monitor >> View Processes. For your production implementation of the code, you might put the Windows or SQL Server login ID into the context information so that you can immediately tell what user is responsible for a given process on the SQL Server instance.

# SET SHOWPLAN

The SET SHOWPLAN statement is an excellent way to reveal the execution plan of a query or transaction. You can also use the graphic showplan option in SSMS to see a query's execution plan, which definitely has its advantages in certain situations. The graphic showplan option in SSMS is especially useful when doing side-by-side comparisons of two queries, as you'll soon see in some examples below. These tools will show you exactly how much processing, as a percentage, was consumed by each step of a batch. The graphic showplan tells you which alternatives are more, or less, costly to the query engine. You can also run two (or more) queries in a single batch and see which one performed the best.

If you just want quick and dirty execution plan details, then use the SET SHOWPLAN\_TEXT variant of the statement. The SET SHOWPLAN\_ALL variant shows a great deal more information about execution plans. And to get the motherload of information about a given query's execution plan, use the SET SHOWPLAN\_XML variant. In this white paper, the examples will primarily use SET SHOWPLAN\_TEXT.

Whichever method you use, there are a few operations that usually indicate inordinate resources consumption or an ineffective plan. Some operations to watch out for include the following:

LOOKUP  
PARALLELISM  
SCANS  
SPOOLS

Remember, these operations are not necessarily bad. Their presence in a query plan may simply indicate that the given operation is the only available means of answering the query. To use an analogy, sports cars are fast, but you'll need a minivan to transport eight people at once. In the same way, there are sometimes faster operations available to the optimizer (such as SEEK compared to SCAN), but sometimes the slower operation is the only method available to accomplish what you want. Nevertheless, when there are multiple ways that the query can be coded, you might opt for strategies that minimize the presence of slower-running operations in favor of other, better-performing alternatives.

## An Execution Plan Does Not Require an Executioner

For some reason, the term "execution plan" invokes images of a burly medieval man with a big ax, clad in a tunic and a dark hood concealing all facial features except for a pair of glowering, pupilless eyes. Fortunately, we're not talking about that sort of execution plan. In SQL Server, an execution plan (also known as a query plan or optimizer plan or—for those who can't shake their Oracle background—the explain plan) is the set of operations that the SQL Server query optimizer must perform to return the result set requested by a given query. You can discover all of a given query's behind-the-scenes operations by using the SET SHOWPLAN\_ALL ON statement presented earlier.

In general, when you examine an execution plan for tuning opportunities, you're looking for a short-list of red flags, such as:

- The query isn't using useful indexes, often because a useful index is missing or the query optimizer somehow overlooks a useful index
- The query isn't using good statistics for an index, because the statistics are out of date (also called "stale statistics") or aren't representative of the distribution of values in the index (that is, the index's cardinality)
- The query isn't using partitions properly
- The transaction is dealing with I/O issues, often the result of a data or index hot spot
- And when we compare two queries against each other, we're often looking for an indication that one version of a query is raising a red flag where another version of the query, which returns the same result set, is not

## Yes, Sir! SARG, Sir!

The single most important construct in a query affecting whether you'll see any of the above-mentioned red flags within the execution plan comes in the form of a search argument, better known as a SARG. A SARG itself come in two forms: as an element of the query's WHERE clause and as an element of the query's JOIN clause. A SARG is a filter. It restricts the result set of a query so that it effectively answers your question. A SARG has a major impact on one of the two ways that the query optimizer is able to figure out the expense of a given query:

1. The number of rows processed at each level of the query, also known as cardinality, and used as an input to the next item (below);
2. The cost of the algorithm according to the kind of operators used in the query.

We know from database design recommendations that we should create indexes on tables to increase the speed with which we can access the data. When we build indexes, we are implicitly improving the cardinality of the parent table. Although there are enough rules and best practices about the creation and use of indexes to fill a separate white paper, you will generally want to create indexes on the columns of a table that are frequently referenced in WHERE clauses and JOIN clauses. As you'll see in later examples within this document, a well-formed SARG (that is, a WHERE clause or JOIN clause) can make or break the performance of a query. Alternately, a bad SARG can cause a query to use a less than optimal index or, in some cases, to completely ignore an existing index.

Currently, the list of standard search argument operators is composed of: =, >, <, =>, <=, BETWEEN and LIKE.

## Which Is Better? Comparing Two Variants as Illustrated by SEEK or SCAN Operations

You'll use execution plan information to compare the relative effectiveness of two separate queries. For example, you might want to see if one query, compared to another, adds extra layers of overhead or chooses a different and non-optimal indexing strategy.

In the previous sections, you were shown a few very important techniques that will help you determine what the overall workload of a query might be. Now, let's take a look at a couple simple variations on a theme for a single query.

One of the first things you'll need to distinguish in an execution plan is the difference between a SEEK and a SCAN operation. A simple but useful rule of thumb is that a SEEK operation performs the best, while a SCAN operation is less than optimal, if not overtly bad. A SEEK goes quickly, via an index, to the affected records, while a SCAN reads the entire object, whether it is a table, a clustered index (which is essentially the entire table) or a non-clustered index. Thus, a SCAN usually consumes a lot more resources than a SEEK. If your execution plan shows SCANS only, then you should consider tuning the query.

In the following examples, we compare two queries, both of which are attempting to return all sales records from the big\_sales table, where the store's ID number is in the 6,300s. The first variant uses the LIKE operator, while the second variant checks for a value of a substring using the SUBSTRING function:

```
SELECT *
FROM big_sales
WHERE stor_id LIKE '63%'
```

The execution plan and statistics I/O returned by the previous query looks like this:

```
--Clustered Index Seek(OBJECT:([big_sales].[UPKCL_big_sales]),
  SEEK:([ big_sales].[stor_id] >= '62b' AND [big_sales].[stor_id] < '64'),
  WHERE:([ big_sales].[stor_id] like '63%') ORDERED FORWARD)
```

Table 'big\_sales'. Scan count 1, logical reads 10, physical reads 0



The query above is able to perform a SEEK for specific values against the clustered index, consuming a very light number of scans and logical reads. The SHOWPLAN describes exactly what the seek operation is based upon—the unique, primary key clustered on the value of the stor\_id column. It also reveals that the results are “like ‘63%’” and ORDERED according to how they are currently stored in the index mentioned. Since SQL Server supports forward and backward scrolling through indexes, with equally good performance for both, you might see either ORDERED FORWARD or ORDERED BACKWARD in the execution plan. This merely tells you which direction the table or index was read. You can even manipulate this behavior by using the ASC and DESC keywords in an ORDER BY clause, if you’ve used one.

Compare this to a query that returns the same result set using a SUBSTRING function:

```
SELECT *
FROM big_sales
WHERE SUBSTRING(stor_id,1,2) = '63'
```

The execution plan and statistics I/O returned by the previous query looks like this:

```
|--Clustered Index Scan(OBJECT:([big_sales].[UPKCL_big_sales]),
  WHERE:(substring([big_sales].[stor_id],(1),(2))='63'))
```

Table 'big\_sales'. Scan count 1, logical reads 79, physical reads 0

Even though both queries retrieved the same result set, the first query with its simple CLUSTERED INDEX SEEK operation is a better execution plan than the second query with its CLUSTERED INDEX SCAN. In this case, the statistics I/O count for the query, using the SUBSTRING function, shows that it consumed **almost eight times as many logical reads** to return the same result set!

A SEEK operation is invoked by either a WHERE clause or a JOIN clause. A SCAN operation, while certainly able to retrieve data, is less efficient.

In the case of the earlier examples using LIKE and SUBSTRING, the predicate tells you what records the WHERE clause filters from the result set. Because this is done as a component of the SEEK or SCAN operation, the WHERE predicate often neither hurts nor helps performance of the operation itself. What the WHERE clause does do is help the query optimizer find the best possible indexes to apply to the query. In the case of the query using SUBSTRING, the function applied to the column stor\_id causes SQL Server to disregard any indexes existing on that column and forces a scan.

Another useful way to compare the queries is to execute them both in a single query window of SSMS while displaying the estimated execution plan (either by choosing Query >> Display Estimated Execution Plan or by right-clicking in the query window and then choosing Display Estimated Execution Plan). When two or more query execution plans are displayed, SQL Server shows the relative cost of each batch as the first line of the report for each query. Thus, by looking at Figure 1 below, we can see that the variant of the query that uses the LIKE comparison operator uses only 12 percent of the total resources consumed by the batch, while the variant of the query that used the SUBSTRING function consumed the lion’s share of the resources used at 88 percent.

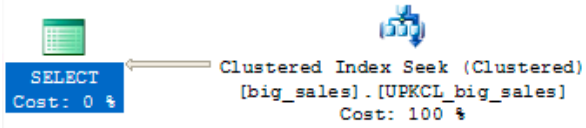
If you’d like to discover more about how indexes make queries perform faster, here are a few articles and posts:

- [http://sqlserverpedia.com/wiki/Index\\_Selectivity\\_and\\_Column\\_Order](http://sqlserverpedia.com/wiki/Index_Selectivity_and_Column_Order)
- <http://sqlserverpedia.com/blog/sql-server-bloggers/when-is-a-seek-actually-a-scan/>
- <http://sqlserverpedia.com/blog/sql-server-bloggers/catch-all-queries/>

---

Query 1: Query cost (relative to the batch): 12%  
--SET STATISTICS IO ON --GO SELECT \* FROM big\_sales WHERE stor\_id LIKE '63%'

---



---

Query 2: Query cost (relative to the batch): 88%  
SELECT \* FROM big\_sales WHERE SUBSTRING(stor\_id,1,2) = '63'

---

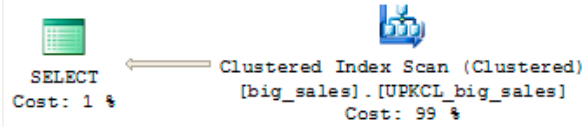


Figure 1: Examining Two Execution Plans Side-By-Side in SSMS



# Special Case Scenarios for Query Tuning

The first part of this white paper showed you a simple, textual method for analyzing the resources consumed by a query, and how that query is processed by the optimizer to return a particular result set. Now, in the following section, we will explore several common query scenarios in which better-performing alternatives are available.

## Functions and Expressions That Suppress Indexes

In many cases, the optimizer cannot use indexes on the columns of a SARG when built-in functions or expressions are applied to an indexed column. The example immediately before this section, in which a SUBSTRING function was applied to an indexed column, illustrates this principle in action. To avoid this problem, try to rewrite these conditions so that index keys are not involved in the expression or built-in function.

In effect, you must help SQL Server by removing any expressions around columns that are indexed. Otherwise, SQL Server may not be able to utilize the index. In some cases, when you apply an expression or function to the column, the index is suppressed. But once you move the expression or function from the column to the value that the SARG is evaluated against, the index is usable once again.

The following queries select a row from a table using a SARG based upon a column which also serves as the unique clustered index. In each case, the column referenced in the WHERE clause is an indexed column. The reason that the index is suppressed is shown as a comment in the code.

Query With Suppressed Index	Optimized Query Using Index
<pre>SELECT * FROM big_sales WHERE SUBSTRING(stor_id,1,2) = '63' -- function on the indexed column</pre>	<pre>SELECT * FROM big_sales WHERE stor_id LIKE '63%'</pre>
<pre>SELECT * FROM big_sales WHERE (stor_id-146) = '7896' -- implicit conversion and expression -- on the indexed column</pre>	<pre>SELECT * FROM big_sales WHERE stor_id = '8042'</pre>
<pre>SELECT * FROM jobs WHERE (job_id + 7) = 14 -- mathematic expression on the -- indexed column</pre>	<pre>SELECT * FROM jobs WHERE job_id = 7</pre>
<pre>DECLARE @job_id VARCHAR(5) SELECT @job_id = '2' SELECT * FROM jobs WHERE CAST(job_id AS VARCHAR(5)) = @job_id -- explicit conversion on the indexed -- column</pre>	<pre>DECLARE @job_id VARCHAR(5) SELECT @job_id = '2' SELECT * FROM jobs WHERE job_id = CAST(@job_id AS SMALLINT)</pre>
<pre>CREATE INDEX employee_hire_date ON employee ( hire_date ) GO -- Get all employees hired -- in the 1st quarter of 1990  SELECT hire_date</pre>	<pre>CREATE INDEX employee_hire_date ON employee ( hire_date ) GO -- Get all employees hired -- in the 1st quarter of 1990  SELECT hire_date</pre>

Query With Suppressed Index	Optimized Query Using Index
<pre> FROM     employee WHERE    DATEPART( year, hire_date )         = 1990         AND DATEPART( quarter, hire_date )         = 1  -- Function on the indexed column </pre>	<pre> FROM     employee WHERE    hire_date &gt;= '1/1/1990'         AND hire_date &lt; '4/1/1990' </pre>

The quick and dirty way to remember this issue when writing a query is that the SARG should place any functions or mathematical expressions not upon a column in the search argument, but upon the value to which it is compared. However, there is a bit more to it than that. The following list details all the situations where SQL Server has difficulty accurately calculating the cardinality of a given query:

1. Queries where the SARG compares values between different columns of the same table
2. Queries where the SARG uses operators with any of these characteristics:
  - a. There are no statistics on the columns involved on either side of the operators
  - b. The query should be a highly selective value set, but it is actually not uniformly distributed, especially if the operator is anything other than an equal sign (=) (for example, an indexed ZIP CODE column that contains millions of records but only a handful are not "90120")
  - c. The SARG uses the not equal to (!=) comparison operator or the NOT logical operator, especially if the column allows NULL
3. Queries that use any of the SQL Server built-in functions or a scalar-valued, user-defined function whose argument is not a constant value (as shown above)
4. Queries that involve joining columns through arithmetic or string concatenation operators (as shown above)
5. Queries that compare variables whose values are not known when the query execution plan is built [i.e. when it is compiled and optimized, for example WHERE CONVERT(INT, my\_column) = @my\_val].

## Head Fakes to the Query Optimizer

Sometimes, when turning a query, you want to force SQL Server to explore other ways to build the execution plan without resorting to a query hint. A common way to achieve this result is to use a function call, such as COALESCE, to force a new query plan. (You could similarly use other functions, such as ISNULL or NULLIF to get the same behavior.)

Normally, COALESCE returns the first nonnull expression from its arguments. But when you enter two arguments, both of which are identical, COALESCE will not affect the result set but will indeed force a different execution plan. For example:

QUERY WITHOUT COALESCE	QUERY WITH COALESCE
<pre> SELECT s1.stor_id FROM sales S1, stores S2 WHERE s1.stor_id = s2.stor_id </pre>	<pre> SELECT s1.stor_id FROM sales S1, stores S2 WHERE s1.stor_id =       COALESCE(s2.stor_id, s2.stor_id) </pre>

The COALESCE transformation usually has one of the following effects on an execution plan: either to rearrange the joining path or to disable internal database transformations.

As described earlier in the section "Functions and Expressions that Suppress Indexes," using the COALESCE function call will stop the index search on s2.stor\_id in our example above.

A similar situation can arise when addressing the transitive property of evaluations. You might recall the transitive property from high school algebra in which we can say “If A = B, and B = C, then A = C”. Well, you’d think that the query optimizer would always be able to apply the transitive property to our queries and thus transform this query:

```
SELECT s1.stor_id
FROM sales S1, stores S2
WHERE s1.stor_id = s2.stor_id
      AND s1.stor_id = 6380
```

Into this query:

```
SELECT s1.stor_id
FROM sales S1, stores S2
WHERE s1.stor_id = s2.stor_id
      AND s1.stor_id = 6380
      AND s2.stor_id = 6380
```

But transitive conversions don’t always happen automatically. Because it’s rather unpredictable as to when the transitive property will be applied behind the scenes by the query optimizer, if it’s applied at all, it’s usually advisable to hard-code the transitive conversions yourself.

Note: using the COALESCE function call or transitive SARGs does not always guarantee better performance. You’ll have to check the performance of each variation of the query explicitly to see which performs best. So, while head faking the query optimizer is not guaranteed to always improve your performance, it will usually alter the execution plan of a query and allow you to try new alternatives.

## Subqueries Optimization

As a good rule of thumb, try to replace subqueries with joins where possible. The optimizer may sometimes automatically flatten out subqueries and replace them with regular or outer joins. But it doesn’t always do a good job at that. Explicit joins give the optimizer more options to choose the order of tables and find the best possible plan. When you optimize a particular query, investigate if getting rid of subqueries makes a difference.

### Example

The following queries select the names of all user tables in the PUBS database and the clustered index name for each table if one exists. If there is no clustered index, then table name still appears in the list with a dash in the clustered index column. Both queries return the same result set, but the first one uses a subquery, while the second employs an outer join. Compare the execution plans produced by Microsoft SQL Server.

Subquery Solution	Join Solution
<pre>SELECT st.stor_name AS 'Store',        ISNULL((SELECT SUM(bs.qty)                FROM sales AS bs                WHERE bs.stor_id = st.stor_id), 0)        AS 'Books Sold' FROM   stores AS st WHERE  st.stor_id IN       (SELECT DISTINCT stor_id        FROM sales)</pre>	<pre>SELECT st.stor_name AS 'Store',        SUM(bs.qty) AS 'Books Sold' FROM   stores AS st JOIN   sales AS bs       ON bs.stor_id = st.stor_id GROUP BY st.stor_name</pre>
<pre>--Compute Scalar (DEFINE: ([Expr1009]=isnull  ([Expr1007], (0))))  --Nested Loops(Left Outer Join,   OUTER REFERENCES: ([st].[stor_id]))  --Nested Loops(Left Semi Join,</pre>	<pre>--Stream Aggregate (GROUP BY: ([st].[stor_name]) DEFINE:  ([Expr1004]=SUM([sales].[qty] as  [bs].[qty])))  --Nested Loops(Inner Join, OUTER   REFERENCES: ([st].[stor_id]))</pre>

<pre> WHERE:([ stores].[stor_id] as [st].[stor_id]=[sales].[stor_id]))     --Clustered Index Scan (OBJECT: ([ stores].[UPK_storeid] AS [st]))     --Clustered Index Scan (OBJECT:([ sales].[UPKCL_sales]))  --Stream Aggregate (DEFINE:([Expr1007]= SUM([sales].[qty] as [bs].[qty])))  --Clustered Index Seek (OBJECT: ([sales].[UPKCL_sales] AS [bs]), SEEK:([bs].[stor_id]= [stores].[stor_id] as [st].[stor_id]) ) </pre>	<pre>  --Sort (ORDER BY:([st].[stor_name] ASC))     --Clustered Index Scan (OBJECT: ([ stores].[UPK_storeid] AS [st]))  --Clustered Index Seek (OBJECT: ([ sales].[UPKCL_sales] AS [bs]), SEEK:([bs].[stor_id]=[ stores].[stor_id] as [st].[stor_id])) </pre>
Table 'sales'. Scan count 7, logical reads 24, physical reads 0, read-ahead reads 0.	Table 'sales'. Scan count 6, logical reads 12, physical reads 0, read-ahead reads 0.
Table 'stores'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.	Table 'stores'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.

Without probing deeper, we see that the join variation of the query required one fewer scan and half as many total logical reads as the subquery solution.

Incidentally, the result sets are the same in both cases, though the sort orders are different because the join query has an implicit ORDER BY clause due to its GROUP BY clause:

```

Store                                     Books Sold
-----
Barnum's                                 154125
Bookbeat                                 518080
Doc-U-Mat: Quality Laundry and Books     581130
Eric the Read Books                       76931
Fricative Bookshop                        259060
News & Brews                              161090

```

(6 row(s) affected)

```

Store                                     Books Sold
-----
Eric the Read Books                       76931
Barnum's                                 154125
News & Brews                              161090
Doc-U-Mat: Quality Laundry and Books     581130
Fricative Bookshop                        259060
Bookbeat                                 518080

```

(6 row(s) affected)

Notice that the execution plan of both queries contains two stream aggregate operations, but the placements of the operations are very different. When an expensive operation is nested within a query so that it must be performed on every iteration of a looping operation, the execution costs can add up quickly. In the examples above, the subquery variant of the query must perform a SUM for each line of the result set that's retrieved. In comparison, the join variant of the query performs the SUM operation as the final, culminating operation of the query.

## UNION vs. UNION ALL

The UNION ALL variant has a number of performance benefits over the more commonly used UNION statement. The difference is that UNION has a side effect of eliminating all duplicate rows and sorting results, which UNION ALL doesn't do.

Selecting a distinct result requires building a temporary worktable, storing all rows in it and sorting before producing the output. (Displaying the showplan on a SELECT DISTINCT query will reveal a stream aggregation is taking place, consuming as much as 30 percent of the resources used to process the query.) In some cases, when that's exactly what you need to do, UNION is your friend. But if you don't expect any duplicate rows in the result set or you don't care about the existence of duplicate records, then use UNION ALL. It simply selects from one query and then mashes subsequent result sets to the bottom of the first result set, as shown by the concatenation operation. UNION ALL requires no work table and no sorting (unless other conditions unrelated to the UNION ALL operator cause the creation of a worktable). One more potential problem with UNION is the danger of flooding tempdb database with a huge worktable. It may happen if you expect a large result set from a UNION query.

The following queries select the ID for all stores in the sales table, which ships as is with the PUBS database, and the ID for all stores in the big\_sales table, a version of the sales table that was populated with more than 10,000 rows. The only difference between the two solutions is the use of UNION versus UNION ALL. But the addition of the ALL keyword makes a big difference in the execution plan.

The first solution requires stream aggregation and sorting the results before they are returned to the client. The second query is much more efficient, especially for large tables.

UNION Solution	UNION ALL Solution
<pre>SELECT stor_id FROM sales UNION SELECT stor_id FROM big_sales</pre>	<pre>SELECT stor_id FROM sales UNION ALL SELECT stor_id FROM big_sales</pre>
<pre> --Merge Join(Union)    --Stream Aggregate(GROUP BY:([sales].     [stor_id]))      --Clustered Index Scan (OBJECT:       ([sales].[UPKCL_sales]))    --Stream Aggregate(GROUP BY:     ([big_sales].[stor_id]))      --Clustered Index Scan(OBJECT:       ([big_sales].[UPKCL_big_sales]))</pre>	<pre> --Concatenation    --Index Scan(OBJECT:([sales].     [titleidind]))    --Index Scan(OBJECT:([big_sales].     [titleidind]))</pre>
<pre>(6 row(s) affected)</pre>	<pre>(10041 row(s) affected)</pre>
<pre>Table 'big_sales'. Scan count 1, logical reads 79, physical reads 0, read-ahead reads 0.</pre>	<pre>Table 'big_sales'. Scan count 1, logical reads 32, physical reads 0, read-ahead reads 0.</pre>
<pre>Table 'sales'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.</pre>	<pre>Table 'sales'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0.</pre>

Although the result sets in this example are largely interchangeable, you can see that the UNION ALL statement consumed less than half of the resources that the UNION statement consumed. So be sure to anticipate your result sets, and in those that are already distinct, use the UNION ALL clause.

## UPDATE...FROM and DELETE...FROM

T-SQL offers an extension to ANSI-SQL syntax for UPDATE and DELETE commands that may be very efficient in many cases. It allows you to specify a FROM clause and join several tables in an UPDATE or DELETE command. It's much easier to read an UPDATE or DELETE statement with a FROM clause. This is because you can easily distinguish the elements of the transaction that filter the UPDATE or DELETE operation upon the affected table (that is, the true WHERE clause) from those elements of the transaction that simply relate the records of the affected table to related tables (that is, the JOIN clause).

### Examples

These principles are illustrated using a few variations of an UPDATE statement. However, these principles apply equally to DELETE statements. To update the titleauthor table, the ANSI SQL solution below executes two correlated subqueries, while the UPDATE...FROM command shown later replaces the subqueries with more explicit joins.

```
UPDATE titleauthor
SET    royaltyper = 90
WHERE  au_id = (SELECT  au_id FROM  authors
                WHERE  au_lname = 'Ringer' AND au_fname = 'Albert')
        AND  title_id = (SELECT  title_id FROM  titles
                WHERE  title = 'Life Without Fear')
```

This yields a rather complex execution plan (edited for brevity) shown here:

```
|--Clustered Index Update(OBJECT:([titleauthor].[UPKCL_taind]),
  SET:([titleauthor].[royaltyper] = 90))
  |--Compute Scalar(DEFINE:([Expr1011]=(90)))
    |--Nested Loops (Inner Join, WHERE:([titleauthor].[title_id]=[Expr1019]))
      |--Assert(WHERE:(CASE WHEN [Expr1018]>(1) THEN (0) ELSE NULL END))
        |--Stream Aggregate(DEFINE:([Expr1018]=Count(*),
          [Expr1019]=ANY([titles].[title_id])))
          |--Index Seek(OBJECT:([titles].[titleind]),
            SEEK:([titles].[title]='Life Without Fear') ORDERED FORWARD)
        |--Nested Loops (Inner Join, OUTER REFERENCES:([Expr1017]))
          |--Assert(WHERE:(CASE WHEN [Expr1016]>(1) THEN (0) ELSE NULL END))
            |--Stream Aggregate(DEFINE:([Expr1016]=Count(*),
              [Expr1017]=ANY([authors].[au_id])))
              |--Index Seek(OBJECT:([authors].[aunmind]),
                SEEK:([authors].[au_lname]='Ringer'
                  AND [authors].[au_fname]='Albert') ORDERED FORWARD)
            |--Index Seek(OBJECT:([titleauthor].[auind]),
              SEEK:([titleauthor].[au_id]=[Expr1017]) ORDERED FORWARD)
```

On the other hand, we can exploit the Transact-SQL extension, allowing the FROM clause and JOIN subclause in the UPDATE statement:

```
UPDATE titleauthor
SET    royaltyper = 90
FROM  titleauthor AS ta
      JOIN authors AS a ON ta.au_id = a.au_id
      JOIN titles AS t ON ta.title_id = t.title_id
WHERE (a.au_lname = 'Ringer' AND a.au_fname = 'Albert')
      AND (t.title = 'Life Without Fear')
```

This yields a simpler execution plan:

```
|--Clustered Index Update(OBJECT:([titleauthor].[UPKCL_taind]),
  SET:([titleauthor].[royaltyper] = [Expr1006]))
  |--Compute Scalar(DEFINE:([Expr1006]=(90)))
    |--Top(ROWCOUNT est 0)
      |--Nested Loops(Inner Join, OUTER REFERENCES:([ta].[title_id]))
        |--Nested Loops(Inner Join, OUTER REFERENCES:([a].[au_id]))
          |   |--Index Seek(OBJECT:([authors].[aunmind] AS [a]),
            |   |   SEEK:([a].[au_lname]='Ringer'
            |   |   AND [a].[au_fname]='Albert') ORDERED FORWARD)
          |   |--Index Seek(OBJECT:([titleauthor].[auind] AS [ta]),
            |   |   SEEK:([ta].[au_id]=[authors].[au_id] as [a].[au_id])
            |   |   ORDERED FORWARD)
        |--Index Seek(OBJECT:([titles].[titleind] AS [t]),
          |   SEEK:([t].[title]='Life Without Fear' AND
          |   |   [t].[title_id]=[titleauthor].[title_id] as [ta].[title_id])
          |   ORDERED FORWARD)
```

In the next example, a row is updated in the titles table that has a specific order recorded in the sales table. Note that the ANSI SQL solution has to execute essentially the same subquery twice, because column title\_id is needed for the WHERE clause and the column qty is used in the SET clause.

ANSI SQL:

```
UPDATE titles
SET ytd_sales = ytd_sales + (
  SELECT qty
  FROM sales s
  WHERE s.stor_id = '7131'
        AND s.ord_num = 'N914014' )
WHERE title_id = (
  SELECT title_id
  FROM sales s
  WHERE s.stor_id = '7131'
        AND s.ord_num = 'N914014' )
```

The execution plan for the ANSI SQL query follows:

```
 |--Clustered Index Update(OBJECT:([titles].[UPKCL_titleidind]),
  SET:([titles].[ytd_sales] = [Expr1009]))
    |--Compute
      Scalar(DEFINE:([Expr1009]=[titles].[ytd_sales]+CONVERT_IMPLICIT(int,[Expr1015],0)))
        |--Nested Loops(Left Outer Join)
          |--Nested Loops(Inner Join, OUTER REFERENCES:([Expr1017]))
            |   |--Assert(WHERE:(CASE WHEN [Expr1016]>(1) THEN (0) ELSE NULL END))
            |   |   |--Stream Aggregate(DEFINE:([Expr1016]=Count(*),
            |   |   |   [Expr1017]=ANY([sales].[title_id] as [s].[title_id]))
            |   |   |--Clustered Index Seek(OBJECT:([sales].[UPKCL_sales] AS
            |   |   |   [s]),
            |   |   |   |   SEEK:([s].[stor_id]='7131' AND
            |   |   |   |   [s].[ord_num]='N914014') ORDERED FORWARD)
            |   |   |--Clustered Index Seek(OBJECT:([titles].[UPKCL_titleidind]),
            |   |   |   SEEK:([titles].[title_id]=[Expr1017]) ORDERED FORWARD)
            |--Assert(WHERE:(CASE WHEN [Expr1014]>(1) THEN (0) ELSE NULL END))
            |   |--Stream Aggregate(DEFINE:([Expr1014]=Count(*),
            |   |   [Expr1015]=ANY([sales].[qty] as [s].[qty]))
```



```

[s]),
|--Clustered Index Seek(OBJECT:([sales].[UPKCL_sales] AS
SEEK:([s].[stor_id]='7131' AND [s].[ord_num]='N914014')
ORDERED FORWARD)

```

Now compare the expansive ANSI SQL update operation shown above and the resultant execution plan with the SQL Server Transact-SQL extension:

```

UPDATE titles
SET ytd_sales = ytd_sales + s.qty
FROM titles AS t
JOIN sales AS s ON t.title_id = s.title_id
WHERE s.stor_id = '7131'
AND s.ord_num = 'N914014'

```

This produces an execution plan with only seven major operations (in comparison, the ANSI SQL plan had eleven):

```

|--Clustered Index Update(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind]),
SET:([pubs].[dbo].[titles].[ytd_sales] = [Expr1004]))
|--Compute Scalar(DEFINE:([Expr1004]=[titles].[ytd_sales] as
[t].[ytd_sales]+[Expr1010]))
|--Top(ROWCOUNT est 0)
|--Nested Loops(Inner Join, OUTER REFERENCES:([s].[title_id]))
|--Compute
Scalar(DEFINE:([Expr1010]=CONVERT_IMPLICIT(int,[sales].[qty] as [s].[qty],0))
|--Clustered Index Seek(OBJECT:([sales].[UPKCL_sales] AS [s]),
SEEK:([s].[stor_id]='7131' AND [s].[ord_num]='N914014')
ORDERED FORWARD)
|--Clustered Index Seek(OBJECT:([titles].[UPKCL_titleidind] AS [t]),
SEEK:([t].[title_id]=[sales].[title_id] as [s].[title_id]) ORDERED
FORWARD)

```

As you might expect, the ANSI SQL query, with its two distinct subqueries, has a larger amount of read activity than that generated by the query containing the FROM...JOIN clause.

## TOP

The TOP clause of the SELECT statement limits the number of rows returned by a single transaction, whether it be a SELECT, INSERT, UPDATE or DELETE statement. This optional clause provides great efficiencies in numerous programming tasks.

Some practical tasks are much more efficient to program with TOP than with standard SQL commands. Let's demonstrate it using several examples. One of the most popular queries in almost any database is a request for the first (that is, the TOP n) items from a long list of selected rows. You could certainly make use of this feature, when returning result sets to a client, to ensure that the application grabs blocks of, say, 30 records at a time. In case of the PUBS database, we could search for the top five best-selling titles. Compare the two solutions—with TOP and using ANSI SQL.

Pure ANSI SQL:

```

SELECT title, ytd_sales
FROM titles a
WHERE ( SELECT COUNT(*)
FROM titles b
WHERE b.ytd_sales >
a.ytd_sales
) < 5
ORDER BY ytd_sales DESC

```

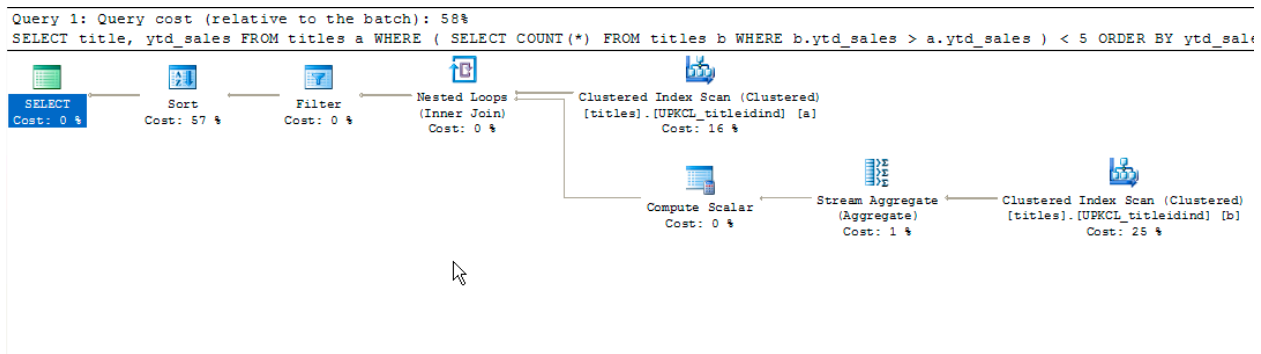


The textual execution plan for the ANSI SQL query looks like this:

StmtText

```
-----  
|--Sort (ORDER BY: ([a].[ytd_sales] DESC))  
  |--Filter (WHERE: ([Expr1004]<(5)))  
    |--Nested Loops (Inner Join, OUTER REFERENCES: ([a].[ytd_sales]))  
      |--Clustered Index  
Scan (OBJECT: ([pubs].[dbo].[titles].[UPKCL_titleidind] AS [a]))  
  |--Compute  
Scalar (DEFINE: ([Expr1004]=CONVERT_IMPLICIT(int, [Expr1008],0))  
  |--Stream Aggregate (DEFINE: ([Expr1008]=Count(*))  
    |--Clustered Index  
Scan (OBJECT: ([pubs].[dbo].[titles].[UPKCL_titleidind] AS  
  [b]), WHERE: ([pubs].[dbo].[titles].[ytd_sales] as  
  [b].[ytd_sales]>[pubs].[dbo].[titles].[ytd_sales] as  
  [a].[ytd_sales]))
```

And the graphic execution plan for the ANSI SQL query is shown below in Figure 2:



**Figure 2: A Graphic Execution Plan in SSMS**

The I/O statistics show that a significant number of reads is needed to complete this query:

Table 'titles'. Scan count 19, logical reads 38, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

The pure ANSI SQL solution executes a correlated subquery which may be inefficient, especially in this case. That's because there is no index on ytd\_sales to support the subquery since the query engine will have to traverse these values one or more times. Additionally, the pure ANSI SQL command does not filter out NULL values in ytd\_sales, nor does it discriminate in the case of a tie between multiple titles.

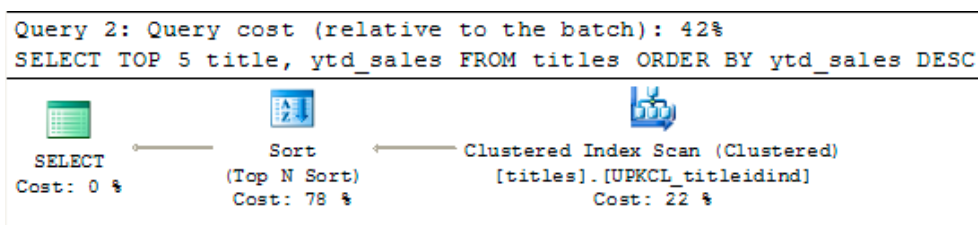
Using TOP n:

```
SELECT TOP 5 title, ytd_sales
FROM titles
ORDER BY ytd_sales DESC
```

The very simple textual execution plan for the TOP query looks like this:

```
StmtText
-----
|--Sort(TOP 5, ORDER BY:([pubs].[dbo].[titles].[ytd_sales] DESC))
  |--Clustered Index Scan(OBJECT:([pubs].[dbo].[titles].[UPKCL_titleidind]))
```

And the graphic execution plan for the TOP query is shown in Figure 3 below:



**Figure 3: A Graphic Execution Plan for a SELECT...TOP Statement**

The I/O statistics show that a very light load of reads, especially when compared to the earlier query, are needed to complete this one:

Table 'titles'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

The second solution, using TOP n to terminate the result set after it has found the first five rows, produces a dramatically simpler and more efficient execution plan. In this case, we also have an ORDER BY clause that forces sorting of the whole table before results may be retrieved. Both queries have almost identical execution plans.

To gain an even greater performance advantage on a large table, we would create an index on `ytd_sales` to avoid sorting. The query would then use the index `SEEK` to find the first five rows and stop, rather than the currently used clustered index `SCAN` looking at all rows in the table. In addition, compare the query plan of the second (`TOP`) query to the first solution (the `ANSI` query), which scans the whole table and executes a correlated subquery for every row retrieved in the outer query. The difference in performance is negligible on a small table. But on a large table, it might amount to hours of processing time for the first solution versus seconds for the second solution.

When determining the needs of your query, consider whether you only need to review a few of the rows retrieved. If that is the case, the `TOP` clause will be a valuable time saver.

## Let's All JOIN Hands and Sing: Understanding the Impact of Joins

If you read through the different query steps earlier in this paper, you saw how a large number of the operations are dedicated to explaining what happens with joins in SQL Server. Every join strategy has its strengths as well as its weaknesses. However, there are certain rare circumstances where the query engine chooses a less efficient join, usually using a hash or merge strategy when a simple nested loop offers better performance.

SQL Server uses three join strategies. They are listed here from the least to the most complex:

### **Nested Loop**

This is the best strategy for small tables with simple inner joins. It works best where one table has relatively few records compared to a second table with a fairly large number of records, and they are both indexed on the joined columns. Nested loop joins require the least I/O and the fewest comparisons.

A nested loop iterates through each record in the outer table once, and then searches the inner table for matches each time to produce the output. There are a lot of names for specific nested loop strategies. For example, a naïve nested loop join occurs when an entire table or index is searched. Other examples include an index nested loop join or a temporary index nested loop join when a regular index or temporary index is used.

### **Merge**

This is the best strategy for large, similarly-sized tables with sorted join columns. Merge operations sort and then cycle through all of the data to produce the output. Good merge join performance is based on having indexes on the proper set of columns, almost always the columns mentioned in the equality clause of the `JOIN` predicate.

Merge joins take advantage of the pre-existing sorts by taking a row from each input and performing a direct comparison. For example, inner joins return records where the join predicates are equal. If they aren't equal, the record with the lower value is discarded and the next record is picked up and compared. This process continues until all records have been examined. Sometimes merge joins are used to compare tables in many-to-many relationships. When that happens, SQL Server uses a temporary table to store rows.

If a `WHERE` clause also exists on a query using a merge join, then the merge join predicate is evaluated first. Then, any records that make it past the merge join predicate are then evaluated by the other predicates in the `WHERE` clause. Microsoft calls this a residual predicate.

### **Hash**

The best strategies for large, dissimilarly sized tables, or for complex join requirements where the join columns are not indexed or sorted is a hashing join. Hashing is used for `UNION`, `INTERSECT`, `INNER`, `LEFT`, `RIGHT`, and `FULL OUTER JOIN`, as well as set matching and difference operations. Hashing is also used for joining tables where no useful indexes exist. Hash operations build a temporary hashing table and then cycle through all of the data to produce the output.

A hash uses a build input (always the smaller table) and a probe input. The hash key (that is, the columns in the join predicate or sometimes in the GROUP BY list) is what the query uses to process the join. A residual predicate is any evaluation in the WHERE clause that does not apply to the join itself. Residual predicates are evaluated after the join predicates. There are several different options that SQL Server may choose when constructing a hash join, in order of precedence:

- **In-memory hash:** This join builds a temporary hash table in memory by first scanning the entire build input into memory. Each record is inserted into a hash bucket based on the hash value computed for the hash key. Next, the probe input is scanned record by record. Each probe input is compared to the corresponding hash bucket and, where a match is found, returned in the result set.
- **Grace hash:** The grace hash option is used when the hash join is too large to be processed in memory. In that case, the whole build input and probe input are read in. They are then pumped out into multiple, temporary work tables in a step called partitioning fan-out. The hash function on the hash keys ensures that all joining records are in the same pair of partitioned worktables. Partition fan-out basically chops two long steps into many small steps that can be processed concurrently. The hash join is then applied to each pair of work tables and any matches are returned in the result set.
- **Hybrid hash:** If the hash is only slightly larger than available memory, SQL Server may combine aspects of the in-memory hash join with the grace hash join in what is called a hybrid hash join.
- **Recursive Hash:** Sometimes the partitioned fan-out tables produced by the grace hash are still so large that they require further re-partitioning. This is called a recursive hash.

**Tip:** Remember that hash and merge joins process through each table just once. Therefore, they might have deceptively low I/O metrics, as does our example in the second query, if you use SET STATISTICS IO ON with queries of this type. However, the low I/O does not mean these join strategies are inherently faster than nested loop joins because of their enormous computational requirements and the fact that they must materialize a worktable in tempDB to complete the processing of the query.

Hash joins, in particular, are computationally expensive. If you find certain queries in a production application consistently using hash joins, this is your clue to tune the query or add indexes to the underlying tables.

In the following example, both a standard nested loop (using the default query plan) and hash and merge joins (forced through the use of hints) are shown:

```
SELECT a.au_fname, a.au_lname, t.title
FROM authors AS a
INNER JOIN titleauthor ta
    ON a.au_id = ta.au_id
INNER JOIN titles t
    ON t.title_id = ta.title_id
ORDER BY au_lname ASC, au_fname ASC
```

StmtText

```
-----
|--Nested Loops (Inner Join, OUTER REFERENCES: ([ta].[title_id]))
  |--Nested Loops (Inner Join, OUTER REFERENCES: ([a].[au_id]))
  |   |--Index Scan (OBJECT: ([pubs].[dbo].[authors].[aunmind] AS [a]), ORDERED FORWARD)
  |   |--Index Seek (OBJECT: ([pubs].[dbo].[titleauthor].[auidind] AS [ta]),
  |       SEEK: ([ta].[au_id]=[a].[au_id]) ORDERED FORWARD)
  |--Clustered Index Seek (OBJECT: ([pubs].[dbo].[titles].[UPKCL_titleidind] AS [t]),
  |   SEEK: ([t].[title_id]=[ta].[title_id]) ORDERED FORWARD)
```

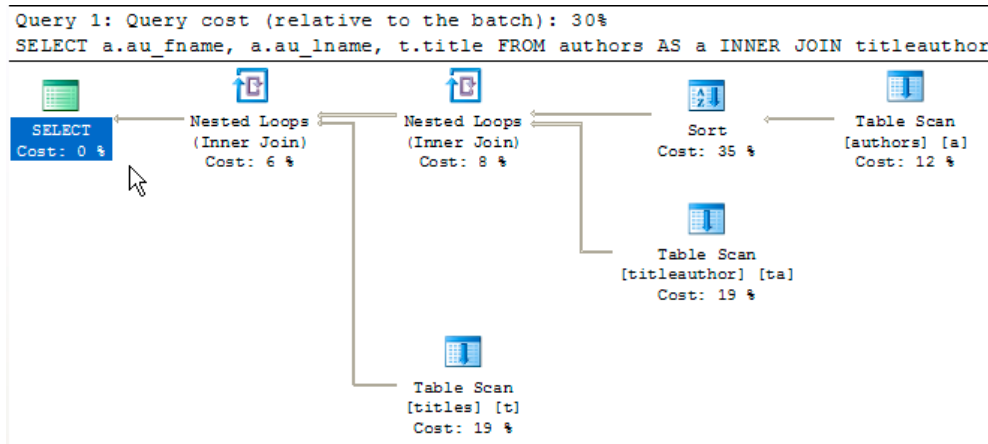
When examining the I/O of this query, we see that a nominal number of reads are performed and that no work table is created to process the result set:

Table 'titles'. Scan count 0, logical reads 50, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'titleauthor'. Scan count 23, logical reads 46, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'authors'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

The visual plan of this query, which consumes the smaller amount of resources at 30 percent compared to the following query, is shown below in Figure 4:



**Figure 4: The Graphic Execution Plan of a Query Using a Nested Loop Join Algorithm**

The showplan displayed above is the standard query plan produced by SQL Server. It can force SQL Server to show you how it would handle these as merge and hash joins using hints:

```
SELECT a.au_fname, a.au_lname, t.title
FROM authors AS a
INNER MERGE JOIN titleauthor ta
    ON a.au_id = ta.au_id
INNER HASH JOIN titles t
    ON t.title_id = ta.title_id
ORDER BY au_lname ASC, au_fname ASC
```

Warning: The join order has been enforced because a local join hint is used.

StmtText

```
-----
|--Sort(ORDER BY:([a].[au_lname] ASC, [a].[au_fname] ASC))
  |--Hash Match(Inner Join, HASH:([ta].[title_id])=([t].[title_id]),
    RESIDUAL:([ta].[title_id]=[t].[title_id]))
    |--Merge Join(Inner Join, MERGE:([a].[au_id])=([ta].[au_id]),
      RESIDUAL:([ta].[au_id]=[a].[au_id]))
      | |--Clustered Index
      | Scan(OBJECT:([pubs].[dbo].[authors].[UPKCL_auind]
        AS [a]), ORDERED FORWARD)
      | |--Index Scan(OBJECT:([pubs].[dbo].[titleauthor].[auind] AS [ta]),
        ORDERED FORWARD)
      |--Index Scan(OBJECT:([pubs].[dbo].[titles].[titleind] AS [t]))
```

Due to the way that hash and merge joins process data, they often exhibit deceptively low I/O:

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

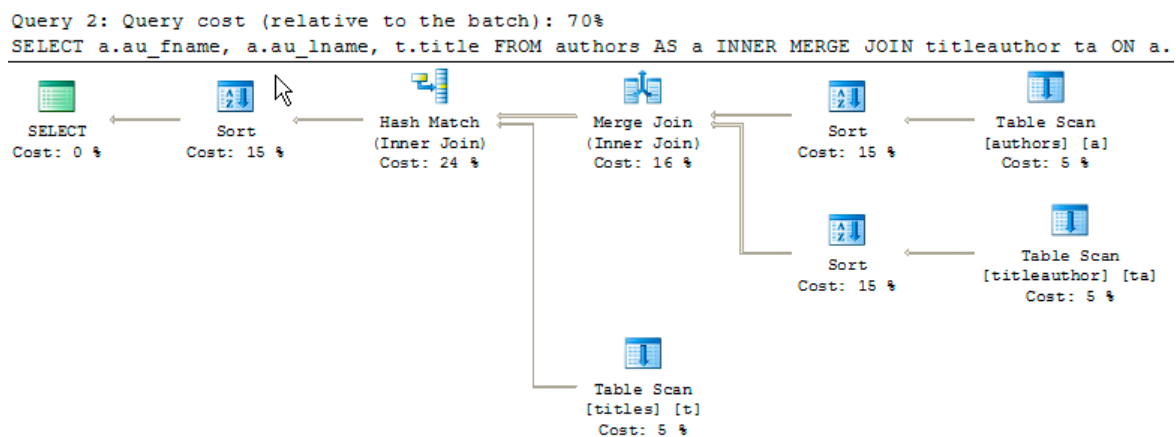
Table 'titles'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'titleauthor'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'authors'. Scan count 1, logical reads 2, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

It's important to note that while the merge and hash joins have lower total scan counts and logical reads than the first query, they also require a worktable in TempDB to process the query. Since the first example—using nested loop joins—does not require a work table, it is able to execute the query with significantly less load.

The visual plan (which shows the hash/merge query, consuming 70 percent of the overall resources as compared to the earlier query's 30 percent), is shown below in Figure 5:



**Figure 5: A Graphic Execution Plan of a Query Using a Hash/Merge Join Algorithm**

In this example, you can clearly see that each join considers the join predicate of the other join to be a residual predicate. (You'll also note that the use of a hint caused SQL Server to issue a warning.) The second query was also forced to use a SORT operation to support the hash and merge joins.

If you'd like to discover more about how to optimize joins, here are a few articles and posts:

- <http://sqlserverpedia.com/blog/sql-server-2005/no-join-predicate/>
- <http://sqlserverpedia.com/blog/sql-server-bloggers/does-order-matter-in-a-join-clause/>
- [http://sqlserverpedia.com/wiki/Optimizer\\_Hints](http://sqlserverpedia.com/wiki/Optimizer_Hints)

## SET NOCOUNT ON

You may have already noticed that, under normal circumstances, successful queries return a system message about the number of rows that they affect. In many cases you don't need this information, especially in procedure code (such as triggers, user-defined functions and stored procedures) that only returns information to the end user via PRINT and RAISERROR statements.

Command SET NOCOUNT ON allows you to suppress the message for all subsequent transactions in your session, until you issue the SET NOCOUNT OFF command. (Yes, it's a double negative, but T-SQL was not created by English majors after all.)

This option has more than a cosmetic effect on the output generated by your script. It reduces the amount of information passed from the server to the client by suppressing the DONE\_IN\_PROC background chatter. Therefore, it helps to lower network traffic and improves the overall response time of your transactions and procedural code. Time to pass a single message may be negligible, but think about a script that executes some queries in a loop and sends kilobytes of useless information to a user.

As an example, consider the following pseudocode to insert 9,999 rows into the sales table:

1. Create some variables
2. Create a loop
3. Assign new variables to an INSERT statement
4. Insert the data
5. Go to step two if loop counter is less than 10,000
6. Print the final running time of the procedural code

When run with SET NOCOUNT OFF, the elapsed time for the example code was 5,176 milliseconds. When run with SET NOCOUNT ON, the elapsed time for the procedural code was 1,620 milliseconds!

Note that the more Transact-SQL commands and/or iterations through the procedural code, the greater the benefit of the SET NOCOUNT ON statement. Consider adding SET NOCOUNT ON at the beginning of every stored procedure and script that doesn't require the "n ROWS AFFECTED" message in the output.

## Querying Against Composite Keys

Composite keys are problematic for SQL Server. Composite indexes are composed of several columns of a table. The problem is that composite indexes are used from the left-most column to right.

The following examples show that SQL Server 2008 now handles poorly ordered WHERE clauses much better than earlier versions of the product. In earlier versions of the product, SQL Server might've ignored indexes when all the columns of an index were addressed in the WHERE clause, solely because the columns were not referenced in the same order as they appeared in the index. This is no longer a problem in SQL Server 2008. However, the problem still impacts how SQL Server chooses execution plans and may cause the optimizer to choose less than optimal plans.

Consider this composite index that contains three columns:

```
CREATE INDEX my_ndx ON new_sales ([stor_id], [ord_num], [title_id] )
```

Depending on your WHERE clause conditions, SQL Server may use all or fewer columns of the index, or not use the index at all, as shown below:

**Table 1. Usage of Composite Key Columns**

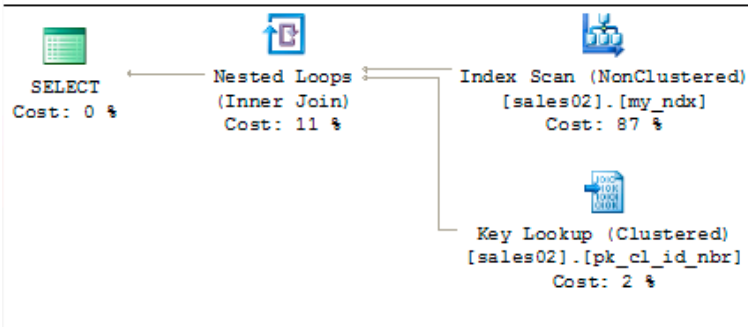
WHERE Clause Conditions	Index Use
WHERE stor_id = @a AND ord_num = @b AND title_id = @c	my_ndx SEEK
WHERE stor_id = @a AND ord_num = @b	my_ndx SEEK
WHERE ord_num = @b AND stor_id = @a	my_ndx SEEK, using the same execution plan as the query above
WHERE stor_id = @a	my_ndx SEEK



WHERE Clause Conditions	Index Use
WHERE stor_id = @a AND title_id = @c	my_ndx SEEK, using the same execution plan as the query above. This query is not able to use the third column of the index.
WHERE ord_num = @b	my_ndx SCAN
WHERE ord_num = @b AND title_id = @c	my_ndx SCAN.
WHERE title_id = @c	my_ndx SCAN.

The key point to remember is that you should know the order of columns appearing within a composite index. Once you know the order of the columns, you should always structure your WHERE clause to analyze columns starting with the left-most column in the composite index and working toward the right. If you build a WHERE clause that does not use leftmost column(s), the index will typically be ignored, resulting in a SCAN operation rather than a better-performing SEEK operation.

Query 6: Query cost (relative to the batch): 20%  
 SELECT qty FROM sales02 WHERE ord\_num = 'P2121'  
 Missing Index (Impact 98.6687): CREATE NONCLUSTERED INDEX [<Name of Missing



Query 7: Query cost (relative to the batch): 22%  
 SELECT qty FROM sales02 WHERE ord\_num = 'P2121' AND title\_id = 'TC7777'  
 Missing Index (Impact 98.7769): CREATE NONCLUSTERED INDEX [<Name of Missing

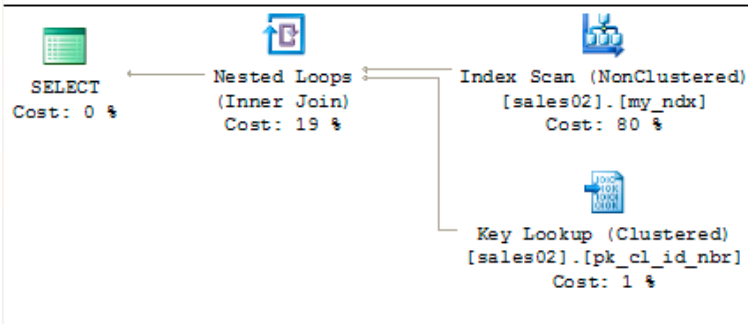


Figure 6: Comparing Two Query Execution Plans, the Second Going Against a Compound Index

As shown above, in Figure 6, when using SQL Server 2008 and the graphic execution plan features of SSMS, you get the added bonus of seeing recommended new indexes that could improve the performance of the query.



# Summary

---

This white paper has presented a collection of tips and tricks to help you get the most out of your queries on a SQL Server 2008 database. Some ideas presented in the white paper include:

- How to use the SET STATISTICS and SET SHOWPLAN commands to see the resource consumption and execution plan of a query
- How to use the DBCC DROPCLEANBUFFERS and DBCC FREEPROCCACHE commands to clear the development server's buffer and procedure cache, using them as elements of your SQL testing harness
- Understanding the basics of reading execution plans, as illustrated by the difference between SEEK and SCAN operations, to determine which variant of a query performs the best
- A variety of scenarios that offer opportunities for performance improvement, such as
  - Functions and expressions that suppress indexes
  - Subquery optimizations versus joins
  - UNION versus UNION ALL
  - The advantages of UPDATE...FROM and DELETE...FROM over ANSI standard syntax
  - TOP and SET ROWCOUNT
  - Alternate join strategies and how they can impact query performance
  - What's so special about SET NOCOUNT ON and why you'd want to use it with procedural code
  - The impact of querying against concatenated keys

Add these tips and techniques to your SQL Server coding toolkit. Be sure to check on the I/O generated by your queries using SET STATISTICS I/O and to read the query execution plans with either SET SHOWPLAN or the graphic execution plans within SQL Server Management Studio. The bottom line is that you need to test, test and retest!

# About the Author

---

**Kevin Kline** is the technical strategy manager for SQL Server Solutions at Quest Software. A Microsoft SQL Server MVP since 2004, Kevin is a founding board member and past president of the international Professional Association for SQL Server (PASS). He has written or co-written several books, including *SQL in a Nutshell*, *Pro SQL Server 2008 Relational Database Design and Implementation*, and *Database Benchmarking: Practical Methods for Oracle & SQL Server*. He is a top rated speaker at conferences worldwide such as Microsoft TechEd, the PASS Community Summit, Microsoft IT Forum, DevTeach, and SQL Connections, and has been active in the IT industry since 1986. Kevin contributes to *SQL Server Magazine* and *Database Trends & Applications* and blogs [SQLBlog.com](http://SQLBlog.com) and [SQLMag.com](http://SQLMag.com).

# Best Practices in Index Maintenance

---

Fighting the Silent Performance Killers

Written by  
Brent Ozar,  
SQL Server DBA

© 2009 Quest Software, Inc.  
**ALL RIGHTS RESERVED.**

This document contains proprietary information protected by copyright. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose without the written permission of Quest Software, Inc. (“Quest”).

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST’S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters  
LEGAL Dept  
5 Polaris Way  
Aliso Viejo, CA 92656  
**www.quest.com**  
email: **legal@quest.com**

Refer to our Web site for regional and international office information.

## Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, ChangeManager, Defender, DeployDirector, Desktop Authority, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, IWatch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, MultSess, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point,Click,Done!, PowerGUI, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, Security Lifecycle Map, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer, vRanger, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Updated—June 2008

# Contents

---

- Abstract.....38
- Index Fragmentation: The Silent (Performance) Killer .....39
  - What Causes Fragmentation? .....39
  - Why is Fragmentation a Problem?.....39
- Maintenance Plans: Not the Solution, but Another Problem .....40
- Hand-Coded Solutions: A Little Knowledge is a Dangerous Thing .....41
- Quest Capacity Manager: A Tested, Feature-Rich Solution.....42
- Conclusion .....44
- About the Author .....45

# Abstract

---

Index fragmentation is a serious issue in every data center because it can cause significant performance degradation. Neither SQL Server's native maintenance plans nor in-house code is an effective solution. However, Quest® Capacity Manager for SQL Server is a tested, feature-rich solution that database administrators (DBAs) can learn quickly and use effectively to defrag indexes and improve database performance.

This document provides a brief introduction to the problem of index fragmentation, explains why native and custom-coded approaches are ineffective, and outlines how Quest Capacity Manager for SQL Server provides a robust and reliable solution.

# Index Fragmentation: The Silent (Performance) Killer

---

Right now in your data center, a vicious, unforgiving villain is slowly draining the performance from your database servers. It strikes without warning and doesn't leave a trace in event logs or messages. This killer is *fragmentation*: data that belongs together logically is not stored together physically. To stop this killer in its tracks, we need two things: an understanding of the warning signs, and a good weapon.

## What Causes Fragmentation?

Let's consider an example. Suppose we have a customer table, and one of our indexes is by Last Name, First Name. Each time we add a customer, one record is added to that index. When the index is first built, all customers whose last name starts with H are together on one page of the index. But as we add additional customers whose last name starts with H, that index page fills up. When the index page is full, SQL Server has to add a new page to the index, and that new page will be added wherever there's space on the disk. Since the index pages are no longer physically together, we have fragmentation.

## Why is Fragmentation a Problem?

Fragmentation degrades database performance in multiple ways. Because SQL Server needs to sort the data in order, index scans and seeks are slower when the data is fragmented. In addition, the new page that was added for H customers is nearly empty; this is known as *low page density*. Reading a nearly empty page off the disk takes just as long as reading a full page, so having many nearly empty pages in a table can cause serious I/O slowdown.

Moreover, if we rely on SQL Server's error logs and warning messages to alert us of high fragmentation levels, we will never see the problem coming.

# Maintenance Plans: Not the Solution, but Another Problem

---

SQL Server does ship with a weapon to fight fragmentation: maintenance plans to defrag indexes. Unfortunately, they can do more harm than good. For starters, they have a start time, but no end time. When the index maintenance job starts, it won't stop on its own, regardless of how long the job takes. If the job continues running after a maintenance window and starts blocking application transactions, our cell phone will ring.

Data warehouses can't use the stock maintenance plans either, because defragmenting hundreds of gigabytes worth of indexes can take an entire weekend. Reindexing a terabyte-size warehouse with a maintenance plan is out of the question.

Moreover, SQL Server's maintenance plans aren't smart enough to look at the fragmentation level of each table and defrag or rebuild it only when required; they touch every table, every time. We can set up the maintenance plan to address only specific individual tables ahead of time, but that doesn't help when the tables have different fragmentation levels every day.



# Hand-Coded Solutions: A Little Knowledge is a Dangerous Thing

---

Given the drawbacks of SQL Server's maintenance plans, many of us choose to write our own T-SQL code to perform index defragmentation. Typically, the code begins as a simple loop that checks the fragmentation level each table in each database and defrags as appropriate.

However, index maintenance is trickier than it looks at first glance. The simple loop often becomes larger and larger as code is added to handle online versus offline rebuilding, different types of indexes, and different versions of SQL Server. We can toss in some code to check for time windows in order to avoid overloading the database server. And suddenly, we have dozens of hours invested in code, but not necessarily in testing or documentation. In fact, we don't have an array of servers and databases to use for testing, so the resulting code is likely to be flawed.

For an example of how tricky a hand-coded index defrag solution can be, check out the comments about one developer's personal code at <http://blogs.digineer.com/blogs/larar/archive/2006/08/16/smart-index-defrag-reindex-for-a-consolidated-sql-server-2005-environment.aspx>.

Moreover, even when a company has a fantastic DBA with plenty of free time to manage indexes, sooner or later, staff turnover raises its ugly head. The new DBA may not understand or trust the last DBA's solution, and therefore may discard it and write new code, which will have its own problems.

The sad fact is that good index management is tougher than it looks, and maybe that's why Microsoft hasn't enhanced SQL Server's maintenance plans beyond the bare basics.

# Quest Capacity Manager: A Tested, Feature-Rich Solution

The solution to index management is to deploy a tested, feature-rich solution that is easy to deploy and so easy to learn that it can survive staff turnover.

Quest Capacity Manager for SQL Server can be deployed in production in less time than it takes us to hand-code a solution (not even counting testing and documentation time). Furthermore, Quest Capacity Manager includes more features than we could build on our own. It's the solution we DBAs (who are not programmers) would build if we could.

Quest Capacity Manager's intuitive management interface, shown in Figure 1, can be picked up by new DBAs without training, so new DBAs can get up to speed quickly.

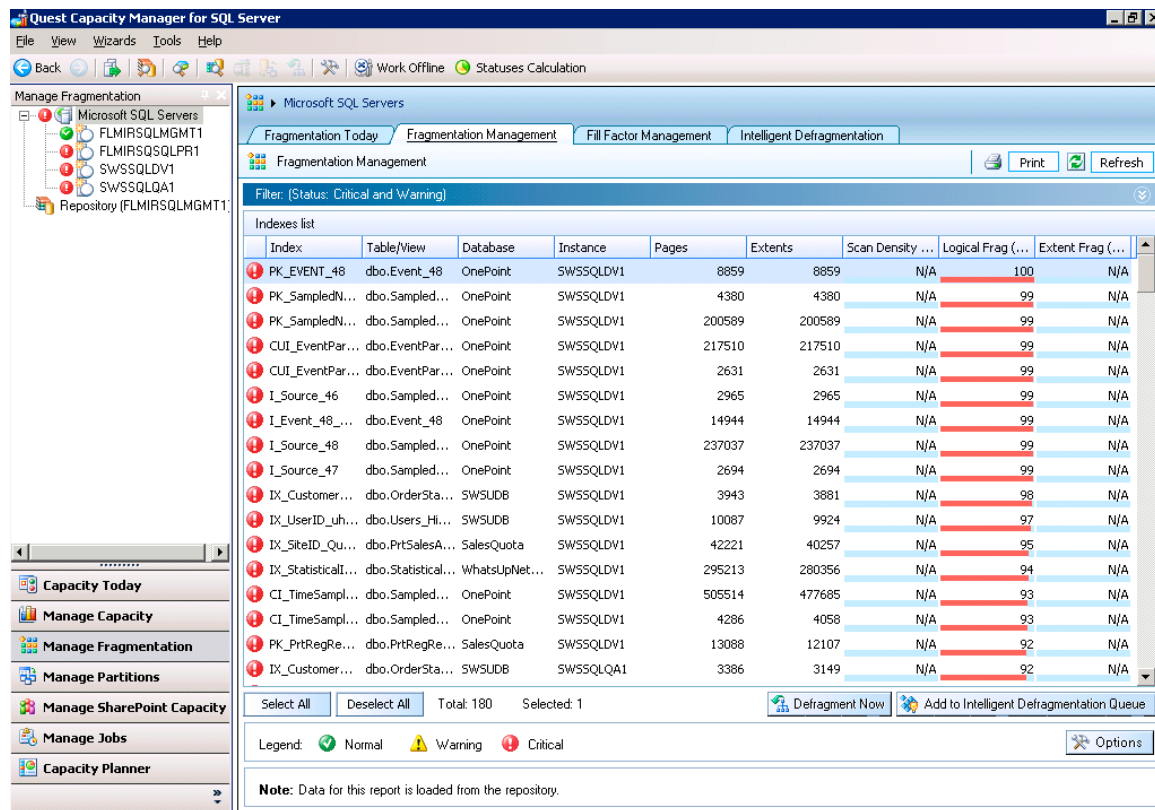
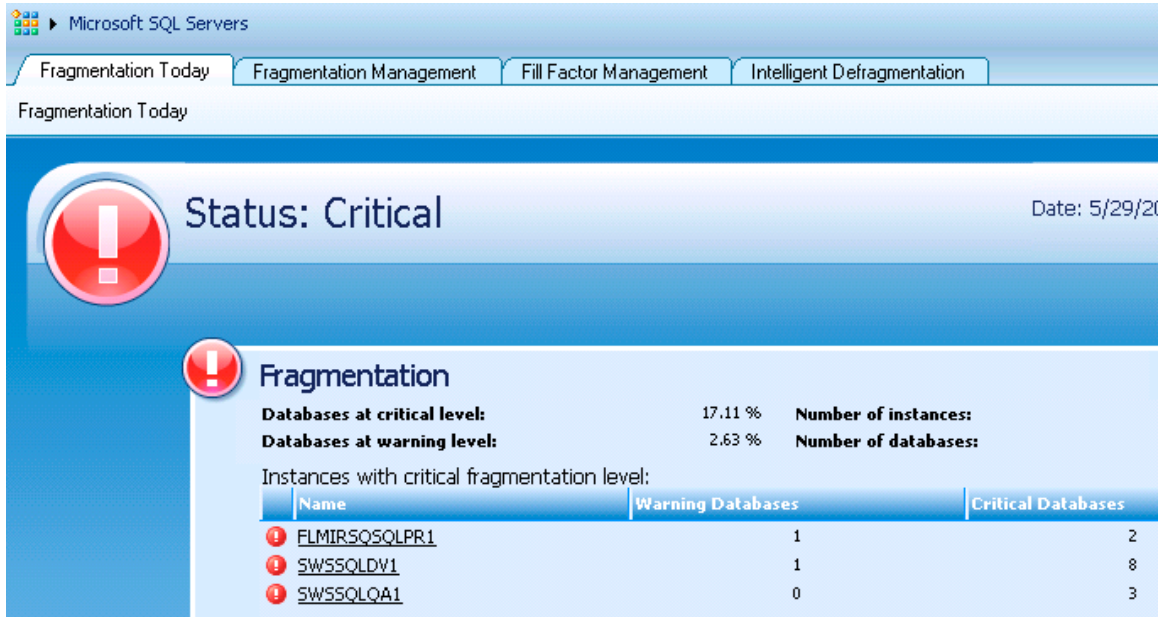


Figure 1: Quest Capacity Manager's Fragmentation Management.

Because no hand-coded solution is involved, staff turnover is less of an issue; less knowledge needs to be transferred from the old DBA to the new one.

Managers can also use Quest Capacity Manager to see how their DBAs are doing with index maintenance. Dashboard-style reports show server health across the enterprise, clearly indicating which servers are in good condition and which suffer from index fragmentation.



**Figure 2: Quest Capacity Manager’s Dashboard-style reports show server health.**

Quest Capacity Manager has been thoroughly tested, both in the QA lab and in real-world production environments. It handles every version of SQL Server and, unlike hand-coded solutions, it doesn’t require hours of our time each time a new version of SQL Server comes out. As each new version arrives, Quest Capacity Manager is ready.

Data warehouses make great candidates for Quest Capacity Manager’s intelligent scheduling. We can set fragmentation thresholds and maintenance windows, and Quest Capacity Manager will do the best possible defragmentation jobs during the time available. Also, it performs online reindexing when possible.

Quest Capacity Manager frees up database administrators to focus on problems beyond regular index maintenance, thereby helping companies to get the best value out of their hard-to-find DBAs. And it helps us successfully combat fragmentation to get the best performance possible.

# Conclusion

---

Every data center needs an effective tool to fight index fragmentation. Unlike SQL Server's native maintenance plans and hand-coded solutions, Quest Capacity Manager for SQL Server is an easy and effective tool for defragging indexes and improving database performance. To learn more about Quest Capacity Manager, visit <http://www.quest.com/capacity-manager-for-sql-server/>.

# About the Author

---

**Brent Ozar** is a SQL Server Expert with Quest Software, and a Microsoft SQL Server MVP. Brent has a decade of broad IT experience, including management of multi-terabyte data warehouses, storage area networks and virtualization. In his current role, Brent specializes in performance tuning, disaster recovery and automating SQL Server management. Previously, Brent spent two years at Southern Wine & Spirits, a Miami-based wine and spirits distributor. He has experience conducting training sessions, has written several technical articles, and blogs prolifically at <http://www.BrentOzar.com>. He is a regular speaker at PASS events, editor-in-chief of SQLServerPedia.com and co-author of the book, "Professional SQL Server 2008 Internals and Troubleshooting."

# Ten Things DBAs Need to Know About Storage

---

Written by  
Brent Ozar,  
SQL Server DBA,  
Quest Software, Inc.

© 2009 Quest Software, Inc.  
**ALL RIGHTS RESERVED.**

This document contains proprietary information protected by copyright. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose without the written permission of Quest Software, Inc. (“Quest”).

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST’S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters  
LEGAL Dept  
5 Polaris Way  
Aliso Viejo, CA 92656  
**www.quest.com**  
email: **legal@quest.com**

Refer to our Web site for regional and international office information.

## Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, ChangeManager, Defender, DeployDirector, Desktop Authority, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, IWatch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, MultSess, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point,Click,Done!, PowerGUI, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, Security Lifecycle Map, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer, vRanger, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Updated— February 2008

# Contents

---

- Introduction .....49
- 1. Know When and How Your Storage Changes .....50
- 2. Backup Times Are the Canary in the Coal Mine.....51
- 3. Investigate with Performance Monitor Counters.....52
  - % Disk Time .....52
  - Average Disk Sec/Read and Sec/Write .....52
  - Average Disk Queue Length .....52
  - Average Disk Read/Sec and Write/Sec .....53
- 4. Build a Relationship with the SAN Team.....54
- 5. Virtualized Storage Can Be Our Worst Enemy.....55
- 6. Virtualized Storage Can Be Our Best Friend .....56
- 7. Shared Spindles Aren't Easily Predictable .....57
- 8. Volume Alignment and Allocation Size Add Speed for Free .....58
- 9. Storage Adapters Can Be Cheap Upgrades .....60
- 10. Stay On Top of New SQL Server Features .....61
- 11. But Wait – There's More! .....62
- Conclusion .....63
- About the Author .....64



# Introduction

---

As a database administrator (DBA) lucky enough to manage my company's SQL Servers as well as the 80 terabyte storage area network, I've fought on both sides of the war. I know why we curse at our SAN administrators, and I know why the SAN administrators throw heavy objects at us.

We are despised for our difficult requests, including our need for the fastest drives on the planet.

SAN administrators, on the other hand, are despised for their sneaky ways: For example, they might really give you a shared set of drives also used for a file server, an e-mail server, and their MP3 collections, thinking you won't know the difference.

But there's a simple way to achieve a happy environment - a place where the DBA and the SAN admin skip down the halls holding hands and the users toss daisies at them from their cubicles. The key is communication. As DBAs, we have to know how to speak the SAN language. Not every DBA has to become a SAN admin the way I did, but we should learn how to tell our SAN administrators exactly what we need, what we're getting, and how we can work together to get there. This paper will dive in to the top 10 things we should know about storage to work effectively with our SAN colleagues.

# 1. Know When and How Your Storage Changes

---

For a DBA, the only indication of a storage change is a performance change: either performance gets better, or it gets worse. In either case, the only way we will know is by constantly monitoring I/O performance across all servers and correlating between them. If all of the servers suddenly get 5% better I/O performance (or heaven forbid, worse), we can probably deduce that a storage change was made.

A performance change may not even last more than a few days. In our shop, one particular SAN firmware upgrade caused the controllers to be more aggressive about marking drives as bad. Prior versions of the firmware allowed a certain number of errors over time without failing the drive, but the new version had a lower tolerance for errors. Drives that had been operating for years were suddenly marked as bad, and the SAN began rebuilding them with hot spares. For the first few days after the firmware upgrade, our performance suffered, but it improved again after the suspect drives were all rebuilt.

The problem is, we're in a war: how do we know when storage changes without relying on the SAN team telling us about them? We need a quick and easy way to know that I/O performance has changed—without running invasive load tests on all of the database servers.

## 2. Backup Times Are the Canary in the Coal Mine

---

Coal miners kept singing canaries around as an early warning system. These birds don't tolerate toxic gases well, so when one died, miners could tell gas levels had approached dangerous levels and get out of the mine shaft while they still had breathable air. Thankfully, DBAs won't die if storage subsystems run out of breathing room, but we still need as much advance warning as we can get.

Full backup jobs put a tough load on the SQL Server's I/O subsystems: they read the entire database from disk storage and write a copy of it somewhere else. Doing a full backup is our best chance to stress test the I/O subsystem without the users complaining—because after all, we will get in trouble if we don't back up the database.

The key to making this process easier is to automate and centralize the reporting data about backups. Quest's LiteSpeed™ for SQL Server brings all backup data into one central repository and provides a simple reporting tool. We can run reports in one place to identify which servers have longer backup times than normal.

Like a dead canary doesn't show the source of deadly fumes in a mine shaft, a sudden increase in backup duration time doesn't pinpoint the exact problem—it's only a basic early warning system. We could have a problem with our read speeds on all drives or on one particular drive. We might also have write speed problems on our backup target, a fast-growing database, a big ETL (extract, transform and load) job that ran into the backup window, or any number of I/O-related issues. But when the canary hits the bottom of the cage, we have to raise an alarm and start testing performance in greater detail.

# 3. Investigate with Performance Monitor Counters

---

When backup times indicate a problem (or even better, do this ahead of time to create a baseline), capture Perfmon counters as a minimally invasive way to measure SQL Server storage performance. The following Physical Disk statistics should be gathered on each individual drive letter on the database server. Capturing the total count might look like a shortcut, but it's misleading because drives with great metrics can cover up a single array with bad metrics.

This information can be captured with Windows' built-in Performance Monitor utility, or with more advanced third-party tools like Quest Spotlight on SQL Server Enterprise or by using server monitoring software like *Ipswitch WhatsUp*. I prefer using the third-party tools because it makes my job easier, but DBAs should understand how to collect and analyze this information with just Perfmon alone. I have an article about using Perfmon up on my site at [www.BrentOzar.com/sql](http://www.BrentOzar.com/sql) that covers it.

## % Disk Time

The % disk time is probably the simplest counter to understand. This number represents the percentage of the time that the array is doing something, so we want it to be as low as possible.

Be aware that arrays with multiple hard drives will go above 100% under heavy loads. The more drives in the array, the higher this number can go, but we want to see it averaging under 50% per drive.

## Average Disk Sec/Read and Sec/Write

These statistics show how many seconds the drives are taking per read or write. Lower numbers indicate faster drives that are able to respond quickly. A measurement of .040 would mean 40 milliseconds.

Anything under .010 is great (that's fewer than 10 milliseconds). Higher numbers should cause concern, and averages of .050 or greater indicate a serious bottleneck. High Sec/Read and Sec/Write numbers can be due to slow drives, an I/O bottleneck like an overloaded SAN switch or slow RAID card, or they could just indicate that we're asking too much from our drives.

When communicating these numbers to the SAN team, we should be sure there's no confusion with a latency issue. SAN administrators measure latency as the amount of time it takes the array to respond to a single request, but they usually measure latency in a vacuum—which shows a very low level of system activity. The Perfmon counter, on the other hand, includes the amount of time the disk request spends just waiting on other requests to finish. If 10 requests get sent simultaneously, the first one will have a very low Sec/Read number, and the tenth one will have a very high Sec/Read number. Since the SAN team members are accustomed to seeing that lower number, they will have a hard time believing the average number.

Numbers higher than .050 aren't necessarily a problem for data warehouses, either. That kind of deep querying is normal for decision support systems. Before sounding an alarm, we need to know whether a lot of queries are waiting on the drives, and how many queries the drives are handling.

## Average Disk Queue Length

This indicates the number of requests waiting on the disk drives trying to get their act together. Ideally, this number should average below two per physical drive in the array. Higher numbers mean that the array in question is a bottleneck.

However, don't micromanage this metric by overanalyzing peaks during high loads. SQL Server does push large batches of I/O at once, and it's not unusual to see peaks in the hundreds or thousands. We're concerned with averages over time, not bursts for seconds at a time. And we're more interested in how the I/O subsystem recovers from these bursts.

## Average Disk Read/Sec and Write/Sec

This captures the number of reads and write sent to the array.

Storage vendors quote ballpark IOPS (input/output operations per second) capacity numbers, but these will be heavily dependent on factors such as the number of physical drives, the raid configuration, the controller cache, and the size of the reads and writes. Because these numbers vary so much by SAN, they are more useful as a relative gauge. They can help us interpret the Sec/Read and Sec/Write metric: as drives get loaded down with more reads and writes, their response times will usually go up. High Sec/Read and Sec/Write numbers should correspond with times of high Read/Sec and Write/Sec numbers.

For help gathering and analyzing these numbers, you can turn to third-party performance products like Quest's Spotlight® on SQL Server. Spotlight continually gathers these numbers (and many more) on the database server in a minimally invasive way. It can produce historical reports as well as a dashboard-style report to instantly pinpoint problems with the I/O subsystems. Even better, you won't need an in-depth understanding of how SQL Server I/O works: that knowledge is built into the product. Spotlight will highlight problem areas in red, and you can drill down to see explanations about why the numbers mean trouble.

Armed with the information captured so far, we can see inside SQL Server to tell the read/write mix on each drive in each server, the amount of requests involved, and which specific arrays are having performance issues. This helps to put us and the SAN team on even footing, and to start speaking the same language.

## 4. Build a Relationship with the SAN Team

---

Before talking to the SAN administrators, we should be aware that they are working with some amazingly advanced hardware. With today's fully redundant SAN controllers and switches, the SAN administrators can upgrade firmware or change configurations on any component of the infrastructure without disrupting the flow of I/O. We can't really stop SAN administrators from tweaking configuration settings or applying new firmware: since it's non-disruptive, they can—and will—do so without telling anyone.

The only way to work with SAN administrators is to make it as easy as possible for them to speed up the database. We have to help them do their jobs better by supporting them in benchmarking the speed of their equipment, and in justifying the tools they need.

For example, whenever we plan a database server maintenance window like a patch, upgrade, or reboot, we should tell the SAN administrators about it. Let them know that the database server will be down for a brief period, and it's their chance to make configuration changes (hopefully improvements) to the server. They may be looking for an opportunity to upgrade the HBA firmware, add additional disks to the arrays, or even migrate one of the arrays to faster drives. Giving them a free outage window makes their job easier and makes them feel like we are looking out for them.

If the SAN team plans on a firmware upgrade or configuration change during the outage window, we need to help them show their success by capturing Perfmon counters before and after the change. A few days after a positive change, we should send them an e-mail recap of the improvements, thanking them for the percentage of decrease in response time or shorter backup windows. We should copy their supervisors on this e-mail as well. This kind of attention to detail shows the SAN administrators that their hard work is appreciated.

To really go the extra mile, we need to give the SAN administrators security permissions so that they can run their own Perfmon tests on the database servers whenever they want. That way, if they make sneaky changes to the SAN, they can go behind our back and see what effect the changes had on the database servers. In a perfect world, they'd work with us hand in hand, but point of our actions here is to promote more open communications between the SAN administrators and our team.

When the SAN administrators understand that we are a partner—and not an obstacle—in the effort to get a fast SAN, they'll open up and be more cooperative. They'll begin to divulge information such as upcoming firmware and configuration changes. If we become performance benchmarking experts, they'll even ask us to help evaluate new SAN gear—such as virtualized storage—as it comes in.

## 5. Virtualized Storage Can Be Our Worst Enemy

---

Virtualized storage allows us to seamlessly move data from fast and expensive raid 10 fiber arrays down to slower, cheaper raid 5 arrays of SATA. Just as SAN administrators can quickly move virtual servers back and forth from fast to slow host servers, they can use virtualized storage to do the same thing for data.

This should strike fear in our hearts because SQL Servers live and die by I/O speed. If SAN administrators are diabolical, they can transparently move a database server's entire data drive without a reboot, without warning, and without our knowing beforehand—and possibly without our knowing afterwards until it's too late.

The most advanced vendor performs this virtual tiering at the block level. A single drive presented to the database could be partially on raid 10 fiber, partially on raid 5 fiber, and partially on SATA. As data is accessed with decreasing frequency, it drops to slower tiers. The telltale sign comes when we do a select against different date ranges of the data. The most current data might perform extremely fast, while data from several months ago performs very slowly.

Another drawback of the virtualized storage option is that the whole SAN design can be undermined by a bad budget. In theory, this virtualized storage technology will automatically push data down to slower tiers of storage only when it's not accessed frequently. However, if SAN administrators don't buy enough fast storage, or buy too much slow storage, the SAN will automatically push some of the blocks down to slower tiers. In that case, all of the server's data lives on the slower storage.

If a shop has virtualized storage, we can never sleep. We must constantly be on the watch, patrolling for sudden I/O bottlenecks, measuring storage statistics night and day.

## 6. Virtualized Storage Can Be Our Best Friend

---

With dangers like this, why would anyone pay money for this kind of storage? If used wisely, virtualized storage offers amazing flexibility.

For example, we usually go to the SAN team well before a new project goes live—but we can't always predict the kind of load an application will have:

- If the application has a slow adoption rate, the SQL Server might be able to live on raid 5 for months
- If the application catches on like wildfire, it might need raid 10, 15 kdrives right away.
- In the worst case scenario, the project never really catches on, and it could live on slow, cheap SATA forever

The DBAs and the SAN teams are responsible for predicting what kind of storage should be purchased. Once it is purchased, the server has to live with it. This type of prediction isn't easy, so we often insist on raid 10 everywhere.

With virtualized storage, the server can start on a relatively slow, cheap array, and move up to a faster disk only if the application shows I/O bottlenecks. Everybody's a winner: the SAN administrators win because they don't waste expensive drives, and we win because we get the speed we need.

Another great example of the flexibility benefit of virtualized storage is in the case of a data warehouse that retains seven years of financial history. The recent data will get heavy read/write access and needs to be on the fastest disk possible. But as time goes on, the data needs to sink down to a slower, cheaper tier of storage. With SQL Server 2005's partitioning, we can automate that movement, but careful planning, testing, and management are required.

This task is performed automatically in block-based virtualized storage. As a particular block is read less and less (relative to the other blocks on the drive), it will be gradually moved between slower tiers, eventually settling on raid 5 SATA. This frees up expensive, fast 15k RPM drives.

Organizations that judiciously use this technology will be the first ones to see ROI potential in solid state drives (SSD). SSDs draw less power and perform faster than even the fastest magnetic hard drives. The drawback is that they cost so much that SSD should be a four letter word. Price aside, they offer unbeatable performance advantages, and this kind of expense can actually be justified in a SAN that uses block-level virtualized storage.

For example, imagine running a data warehouse where today's data loads are written first to screaming fast SSD arrays, and where the morning reports hit those same high-speed flash drives. A few days later, the data sinks down to slower raid 10 FC drives, which are still blazing fast. While it's hard to get an ROI on an entire data warehouse on SSD, this kind of block-level speed is obviously very attractive.

Quest's Spotlight on SQL Server Enterprise helps us and the SAN administrators work together to monitor server I/O performance. With Spotlight Enterprise, a single monitoring server tracks the health of all database servers around the clock. When team members want to check a server's health, they connect their workstations to the monitoring server. That way, numerous people can stay on top of server performance without all of them running invasive monitoring tools on their desktops. Just one instance monitors the SQL Server to gather statistics, and then the users hit that data repository on the Spotlight server.



## 7. Shared Spindles Aren't Easily Predictable

---

SANs can be configured to use one giant pool of drives (or several pools), and then to carve that space up into chunks for each application. A SQL Server might get 100 GBs of space, but that space would be on dozens of different drives shared by many applications running on several servers.

In theory, this set up allows the SAN administrators to spread load more evenly across more drives. And again, in theory, the SQL Server can leverage those additional drives to get better performance under times of heavy load.

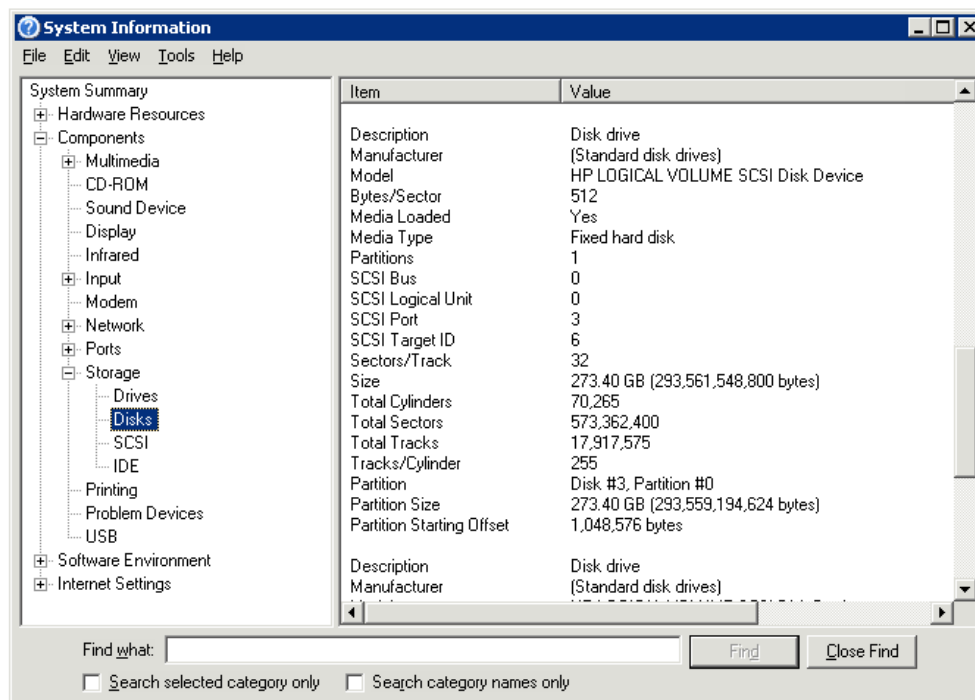
The drawback is that the SQL Server's performance can vary dramatically depending on whether the other applications are also under heavy load. This is one instance where my inner DBA wins out over my inner SAN administrator. The unpredictable performance drops take precedence over any performance gains. I'd rather have consistent performance than performance that goes all over the chart—and shared spindles don't offer consistent performance.

However, we won't be able to win this battle in every shop. If the company's strategy is to share spindles for all applications, then we need to be the squeaky wheel. We should insist that all new applications on the SAN should trigger purchases of additional drive spindles to offset their new load. Ask the SAN team to give reports on which applications are producing the most I/O loads, and ask if the SQL Servers can be moved into their own drive pools.

## 8. Volume Alignment and Allocation Size Add Speed for Free

Allocation unit size and sector/volume alignment are simple configuration parameters used when first setting up a volume. The good news is that they can provide free performance gains (as much as 30%), but the bad news is that they can't be changed once they're set up. The only way to fix these is to blow away the NT file system volume, format it, and restore the data from backup. Cache settings, HBA (host bus adaptor) queue depths, drivers, and other environmental factors can be changed with a reboot or a fast outage. But once the drive formats are set in stone, the only way they can be changed is with a time-consuming backup, disk wipe and a complete restore. Therefore, it's important to get these right initially, and of course, Windows doesn't choose the right settings for SQL Server by default.

Volume alignment, also called partition offset, is set up behind the scenes before a volume is even formatted. To find this disk information on a server that already exists, use the Windows utility MSINFO32. From the server's desktop, click Start, Run, type MSINFO32 and hit enter. Drill down into Components, Storage, Disk. If the Partition Starting Offset is 1,048,576, the storage is aligned, as in the screenshot below:



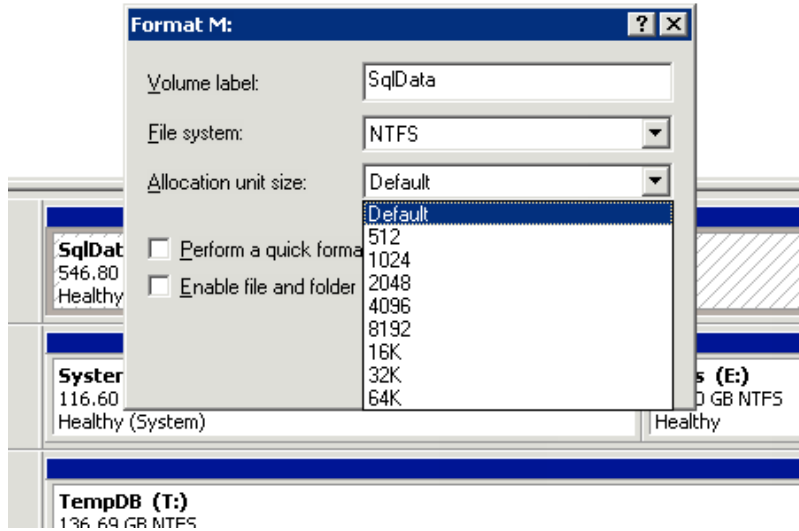
Microsoft's KB articles #929491 and #923076 recommend a Partition Starting Offset of 1,024kb (1,048,576 bytes, or 2,048 512k-byte sectors) because that evenly divides out for almost all SAN configurations.

Other offsets don't necessarily mean the volume isn't aligned. Hard-core storage gurus can pick smaller numbers if they know the exact configuration of the underlying SAN hardware. However, even in those instances, we should use 1,024kb because this gives the SAN team the flexibility to change segment sizes on the fly without worrying about re-aligning the NTFS volumes.

If a volume doesn't show 1,048,576 bytes, it probably needs to be blown away and recreated from scratch. After deleting the volume, use the DISKPART utility as explained in the Microsoft KB article #929491. Windows Server 2008 will do this by default when creating new volumes, but if you're using Windows Server 2003 and previous versions, you will need to do this manually.

After the partition is properly aligned, it's time to set the allocation unit size. When formatting a new volume inside Windows, it's a drop-down option shown in the image below:

Partition	Basic	NTFS	Healthy (System)	116.60 GB	45.46 GB	38 %	No
Partition	Basic	NTFS	Healthy	136.69 GB	8.63 GB	6 %	No



For SQL Server, most storage vendors recommend the largest allocation unit size possible, which for NTFS means 64K. Sadly, this is not the default, so we need to pay attention when we create new volumes.

## 9. Storage Adapters Can Be Cheap Upgrades

---

Users with a dial-up connection to the Internet might believe that their computers are slow, and that they need to spend thousands of dollars on a new computer to get faster web browsing. In fact, the problem is their internet connection speed, and even users with pretty slow computers can have a pleasant experience surfing the web just by spending a little more on their internet connection method.

Storage throughput works that same way: sometimes the bottleneck is the connection between the server and the storage:

- For direct attached storage, that means the RAID cards
- For fiberoptic SAN storage, that means the Host Bus Adapters
- For iSCSI storage, that means the iSCSI network cards

Measuring this throughput is further complicated by the fact that different vendors deliver their adapter metrics in different ways. There's no silver bullet for this one: it requires careful research of the vendors' manuals.

We should watch these metrics during the SQL Server's backup window to see if there's a bottleneck. Backups read the entire database, and drives usually give their best performance during reads. If the storage adapter's bandwidth is over 90% during a backup, then the storage adapter is probably the speed limit. Adding additional storage adapters with load-balancing software can mitigate this problem.

When doubling the number of connections to the SAN, we need to be aware that throughput won't always double. We are sometimes surprised by a dirty secret in the SAN industry: entry-level and mid-range SANs may allow for multiple connections to the SAN, as with multiple HBAs or iSCSI network cards, but they rarely allow true active-active connections. Specifically, they usually present one array (one drive letter) through only one path at a time. The way around this is to divide the database up into several arrays via partitioning or by multiple data files.

# 10. Stay On Top of New SQL Server Features

---

Microsoft SQL Server 2005 introduced Instant File Initialization and partitioning. These two features dramatically improved specific performance areas. If you were one of the savvy DBAs who implemented these features, you got I/O performance gains without spending a dime.

SQL 2008's native data compression and log stream compression offer more storage subsystem performance gains. This time, there's a CPU tradeoff: compression and decompression requires CPU overhead. In the age of quad-core CPUs, and with SQL Server still licensed by the socket (instead of by the core), there's a return on investment for replacing processors in order to gain expensive I/O performance.

Quest's Spotlight helps analyze the server to determine whether it's a good match for SQL 2008's compression features. If CPU load is rarely a bottleneck on the server, but I/O is a bottleneck, then it's probably a good match for SQL 2008's native compression. We can enable this compression one database at a time and then watch the load with Spotlight over time to make sure the server isn't being overloaded.

Looking back, SQL 2005 SP2 brought the VARDECIMAL field type, a new way to store large decimals. This has a large impact on SAP BI/BW databases: Microsoft measured 20-50% size decreases on fact tables during its testing. When table size drops, then I/O load also decreases, because less data needs to be read back during queries. This feature didn't get much attention, but the fact that it came in a service pack tells us that we need to stay current even between releases.

# 11. But Wait – There’s More!

---

The primary reason I made database administration my career, is that I’d never have to learn another programming language—T-SQL rarely changes, and it’s somewhat similar between different platforms, too. While I haven’t had to learn a new language from scratch, I’ve had to keep up with the server industry, the storage industry, and SQL Server’s own growing feature list. We may not have to learn a new programming language, but we have to learn the language of storage area networks, of IP networking, and benchmarking.

# Conclusion

---

The most important thing for us to know about storage is that we're not alone in this struggle. Communicate with each other, with the SAN teams, with other organization in your area of similar size, with the storage vendors, and with Microsoft. Many people you will encounter in these places have entire web sites, blogs and forums devoted to helping peers work with their storage gear and with SQL Server. Reach out, and you'll be surprised who's there to help you!

# About the Author

---

**Brent Ozar** is a SQL Server Expert with Quest Software, and a Microsoft SQL Server MVP. Brent has a decade of broad IT experience, including management of multi-terabyte data warehouses, storage area networks and virtualization. In his current role, Brent specializes in performance tuning, disaster recovery and automating SQL Server management. Previously, Brent spent two years at Southern Wine & Spirits, a Miami-based wine and spirits distributor. He has experience conducting training sessions, has written several technical articles, and blogs prolifically at <http://www.BrentOzar.com>. He is a regular speaker at PASS events, editor-in-chief of SQLServerPedia.com and co-author of the book, "Professional SQL Server 2008 Internals and Troubleshooting."



# Top 10 Things You Should Know About Optimizing SQL Server Performance

---

Written by  
Patrick O'Keeffe,  
Senior Software Architect,  
Quest Software, Inc.

© 2009 Quest Software, Inc.  
**ALL RIGHTS RESERVED.**

This document contains proprietary information protected by copyright. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose without the written permission of Quest Software, Inc. (“Quest”).

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST’S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters  
LEGAL Dept  
5 Polaris Way  
Aliso Viejo, CA 92656  
**www.quest.com**  
email: **legal@quest.com**

Refer to our Web site for regional and international office information.

## Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, ChangeManager, Defender, DeployDirector, Desktop Authority, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, IWatch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, MultSess, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point,Click,Done!, PowerGUI, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, Security Lifecycle Map, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer, vRanger, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Updated— September 2007

# Contents

---

- Introduction .....68
- What Problem Are We Trying to Solve?.....69
- 10. Baselineing and Benchmarking Are a Means to an End .....70
  - What Are Baselineing and Benchmarking? .....70
  - What Baselineing Can't Do.....70
- 9. Performance Counters – How to Cut to the Chase .....72
  - Operational Monitoring .....72
  - Bottleneck Monitoring .....73
- 8. Why Changing sp\_configure Settings Probably Won't Help.....74
- 7. I Have a Bottleneck – What Do I Do Now? .....75
- 6. SQL Profiler is Your Friend .....76
  - Start a Trace.....76
- 5. Zen and the Art of Negotiating with Your SAN Administrator .....80
- 2. The Mystery of the Buffer Cache.....84
- 1. The Tao of Indexes .....86
  - sys.dm\_db\_index\_operational\_stats .....86
  - sys.dm\_db\_index\_usage\_stats .....87
- And an Extra One for Good Measure. Learn XPath .....88
- Conclusion .....90
- About the Author .....91
- Notes.....92

# Introduction

---

Performance optimization on SQL Server is difficult. A vast array of information exists on how to address performance problems in general. However, there is not much information on the specifics and even less information on how to apply that specific knowledge to your own environment.

In this whitepaper, I discuss the 10 things that I think you should know about SQL Server performance. Each item is a nugget of practical knowledge that can be immediately applied to your environment.

# What Problem Are We Trying to Solve?

---

The problem is a simple one: How do you get the most value from your SQL Server deployments? Faced with this problem, many of us ask: Am I getting the best efficiency? Will my application scale?

A scalable system is one in which the demands on the database server increase in a predictable and reasonable manner. For instance, doubling the transaction rate might cause a doubling in demand on the database server, but a quadrupling of demand could well result in the system failing to cope.

Increasing the efficiency of database servers frees up system resources for other tasks, such as business reports or ad-hoc queries.

To get the most out of an organization's hardware investment, you need to ensure that the SQL or application workload running on the database servers is executing as fast and as efficiently as possible.

There are a number of drivers for performance optimization, including:

- Tuning to meet service level agreement (SLA) targets
- Tuning to improve efficiency, thereby freeing up resources for other purposes
- Tuning to ensure scalability, thereby helping to maintain SLAs into the future

Performance optimization is an ongoing process. For instance, when you tune for SLA targets, you can be 'finished'; however, if you are tuning to improve efficiency or to ensure scalability, your work is never really finished. This tuning should be continued until the performance is 'good enough'. In the future, when the performance of the application is no longer 'good enough', this tuning should be performed again.

'Good enough' is usually defined by business imperatives, such as SLAs or system throughput requirements. Beyond these requirements, you should be motivated to maximize the scalability and efficiency of all database servers—even if business requirements are currently being met.

As stated in the introduction, performance optimization on SQL Server is challenging. There is a wealth of generalized information on various data points (counters, DMVs) available, but there is very little information on what to do with this data and how to interpret it. This paper describes 10 things that will be useful in the trenches, allowing you to turn some of the data into information.

# 10. Baselineing and Benchmarking Are a Means to an End

---

## What Are Baselineing and Benchmarking?

Baselineing and benchmarking give you a picture of resource consumption over time. If your application has not yet been deployed into production, you need to run a simulation. This can be achieved by:

- Observing the application in real time in a test environment
- Playing back a recording of the application executing in real time

The best outcome is achieved by observing actual workload in real time or playing back a recording of a real time simulation.

Ideally, you would also want to run the workload on hardware comparable to what the application will be deployed on and with “realistic” data volumes. SQL statements that deliver good performance on small tables often degrade dramatically as data volumes increase. The resulting data can then be plotted to easily identify trends.

The practical upshot is that you can evaluate future behavior against a baseline, to determine whether resource consumption has improved or worsened over time.

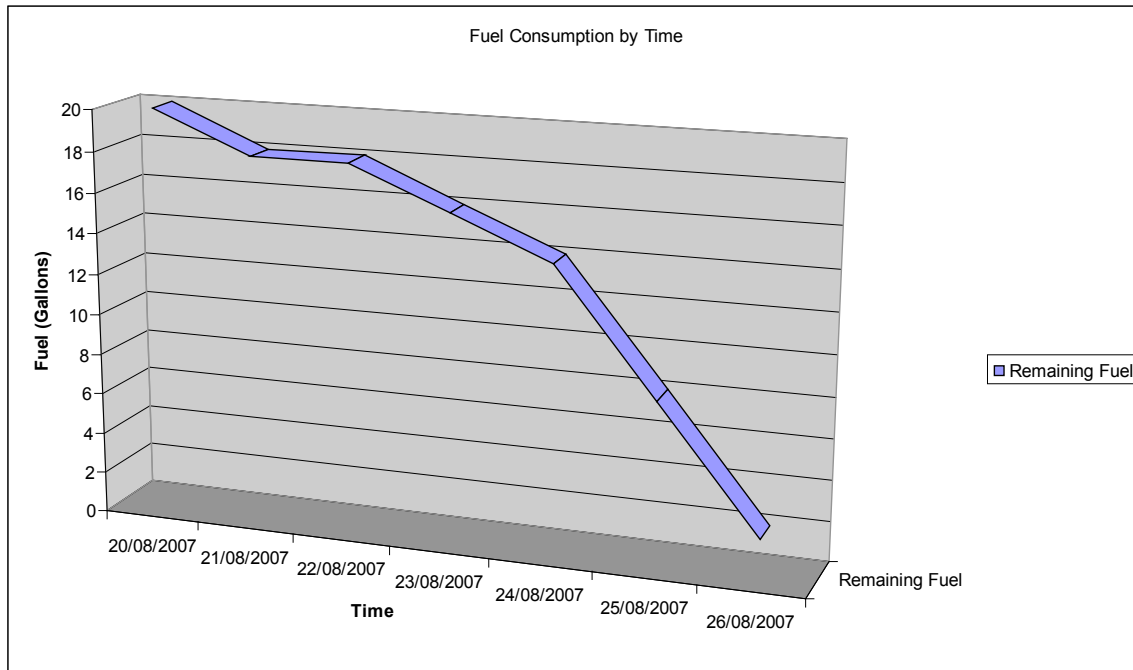
## What Baselineing Can't Do

Baselineing is not the only tool in your performance optimization toolbox. To explain what baselineing and benchmarking can't do, let's use the ubiquitous car analogy and talk about fuel consumption.

The performance counter we are going to sample is obviously a fuel gauge.



For the period of a few days, we will sample the level of the fuel in the fuel tank and plot it in a graph shown below.



The plot displays the fuel remaining in the fuel tank over time. We can see that the baseline behavior represents the level of fuel in the tank that decreases slowly at first and then starts to accelerate more quickly towards the end of the measured time period. In general, this is the normal behavior for fuel in a fuel tank over time.

Assuming this graph represents normal behavior, we can measure and plot a different behavior and compare the two graphs. We would easily see the change in behavior. Emerging trends may also be easily identified since we can plot against time.

A baseline cannot, however, provide any qualitative measure of efficiency. From the chart above, you cannot draw any conclusions about how efficient the car is—you must investigate elsewhere for this information. The baseline can tell you only whether you used more (or less) fuel between two days.

Similarly for SQL Server, a baseline can tell you only that something is outside that range of normally observed behavior. It cannot tell you whether the server is running as efficiently as possible.

The point is that you should not start with baselining. You need to make sure that your application workload is running as efficiently as possible. Once performance has been optimized, you can then take a baseline. Also, you cannot simply stop with baselining. You should keep our application running as efficiently as possible and use your baseline as an early warning system that can alert you when performance starts to degrade.

# 9. Performance Counters – How to Cut to the Chase

A very common question related to SQL Server performance optimization is: What counters should I monitor?

In terms of managing SQL Server, there are two broad reasons for monitoring performance counters:

- Operational
- Bottlenecks

Although they have some overlap, these two reasons allow you to easily choose a number of data points to monitor.

## Operational Monitoring

Operational monitoring checks for general resource usage. It helps answer questions like:

- Is the server about to run out of resources like CPU, disk space or memory?
- Are the data files able to grow?
- Do fixed size data files have enough free space for data?

You also could collect data for trending purposes. A good example would be collecting the sizes of all the data files. From this information, you could trend the data file growth rates. This would allow you to more easily forecast what resource requirements you might have in the future.

To answer the three questions posed above, you should look at the following counters:

Counter	Reason
Processor\% Processor Time	Monitor the CPU consumption on the server
LogicalDisk\Free Megabytes	Monitor the free space on the disk(s)
MSSQL\$Instance:Databases\Data File(s) Size (KB)	Trend growth over time
Memory\Pages/sec	Check for paging, a good indication that memory resources might be short



# Bottleneck Monitoring

Bottleneck monitoring focuses more on performance-related matters. The data you collect helps answer questions such as:

- Is there a CPU bottleneck?
- Is there an I/O bottleneck?
- Are the major SQL Server subsystems, such as the Buffer Cache and Procedure Cache, healthy?
- Do we have contention in the database?

To answer these questions, we would look at the following counters:

Counter	Reason
Processor\% Processor Time	Monitor CPU consumption allows us to check for a bottleneck on the server (indicated by high sustained usage).
High percentage of Signal Wait	Signal wait is the time a worker spends waiting for CPU time after it has finished waiting on something else (e.g., a lock, a latch or some other wait). Time spent waiting on the CPU is indicative of a CPU bottleneck.  Signal wait data can be found by executing <code>DBCC SQLPERF(waitstats)</code> on SQL Server 2000 or by querying <code>sys.dm_os_wait_stats</code> on SQL Server 2005.
PhysicalDisk\Avg. Disk Queue Length	Check for disk bottlenecks – if the value exceeds 2 <sup>1</sup> then it is likely that a disk bottleneck exists.
MSSQL\$Instance:Buffer Manager\Page Life Expectancy	Page Life Expectancy is the number of seconds a page stays in the buffer cache. A low number indicates that pages are being evicted without spending much time in the cache, thereby reducing the effectiveness of the cache.
MSSQL\$Instance:Plan Cache\Cache Hit Ratio	A low Plan Cache hit ratio means that plans are not being reused.
MSSQL\$Instance:General Statistics\Processes Blocked	Long blocks indicate a contention for resources.

## 8. Why Changing sp\_configure Settings Probably Won't Help

---

SQL Server is not like other databases. Very few switches and knobs are available to tweak performance. There are certainly no magic silver bullets to solve performance problems simply by changing an sp\_configure setting.

It is generally best to leave the sp\_configure settings at their defaults, thereby letting SQL Server manage things. Your time is best spent looking at performance from a workload perspective, such as database design, application interaction and indexing issues.

Let's look at a workload example at a setting and see why it is generally best to leave things alone.

The "max worker threads" setting is used to govern how many threads SQL Server will use. The default value (in SQL Server 2005 on commodity hardware) is 256 worker threads.

This does not mean that SQL Server can have only 256 connections. On the contrary, SQL Server can service thousands of connections using up to the maximum number of worker threads.

If you were responsible for a SQL Server that regularly had 300 users connected, you might be tempted to raise the maximum number of worker threads to 300. You might think that having one thread per user would result in better performance. This is incorrect. Raising this number to 300 does two things:

1. Increases the amount of memory that SQL Server uses. Even worse, it decreases the amount of memory that SQL Server can use for buffer cache, because each thread needs a stack
2. Increases the context switching overhead that exists in all multithreaded software

In all likelihood, raising the maximum number of worker threads to 300 made things worse. It also pays to remember that even in a four-processor box, there can only be four threads running at any given time. Unless you are directed to do so by Microsoft support, it is best to focus your efforts on index tuning and resolving application contention issues.

## 7. I Have a Bottleneck – What Do I Do Now?

Once you have identified a bottleneck and worked out that it is best to leave the `sp_configure` settings alone, you need to find the workload that is causing the bottleneck.

This is a lot easier to do in SQL Server 2005. Users of SQL Server 2000 will have to be content with using Profiler or Trace (more on that in #6).

In SQL Server 2005, if you identified a CPU bottleneck, the first thing that you would want to do is to get the top CPU consumers on the server. This is a very simple query on `sys.dm_exec_query_stats`:

```
select top 50
    qs.total_worker_time / execution_count as avg_worker_time,
    substring(st.text, (qs.statement_start_offset/2)+1,
        ((case qs.statement_end_offset
            when -1 then datalength(st.text)
            else qs.statement_end_offset
        end - qs.statement_start_offset)/2) + 1) as statement_text,
    *
from
    sys.dm_exec_query_stats as qs
    cross apply sys.dm_exec_sql_text(qs.sql_handle) as st
order by
    avg_worker_time desc
```

The really useful part of this query is your ability to use `cross apply` and `sys.dm_exec_sql_text` to get the SQL statement so you can analyze it.

It is a similar story for an I/O bottleneck:

```
select top 50
    (total_logical_reads + total_logical_writes) as total_logical_io,
    (total_logical_reads/execution_count) as avg_logical_reads,
    (total_logical_writes/execution_count) as avg_logical_writes,
    (total_physical_reads/execution_count) as avg_phys_reads,
    substring(st.text, (qs.statement_start_offset/2)+1,
        ((case qs.statement_end_offset
            when -1 then datalength(st.text)
            else qs.statement_end_offset
        end - qs.statement_start_offset)/2) + 1) as statement_text,
    *
from
    sys.dm_exec_query_stats as qs
    cross apply sys.dm_exec_sql_text(qs.sql_handle) as st
order by
    total_logical_io desc
```

## 6. SQL Profiler is Your Friend

If we have SQL Server 2000 (which means no DMVs to make your life easier), you can still obtain (with a little more work) similar information to identify and classify workload.

In the following step, I use Profiler Traces from a remote machine. There is nothing to stop you from using server side traces instead. The important part is what you do with the raw data once it gets into a database table.

### Start a Trace

The goal is to classify workload so I have chosen these four SQL-related events:

- RPC:Completed
- SP:Completed
- SQL:BatchCompleted
- SQL:StmtCompleted

Figure 1 shows the Trace Properties Dialog. I have also chosen all possible columns for each of these event types.

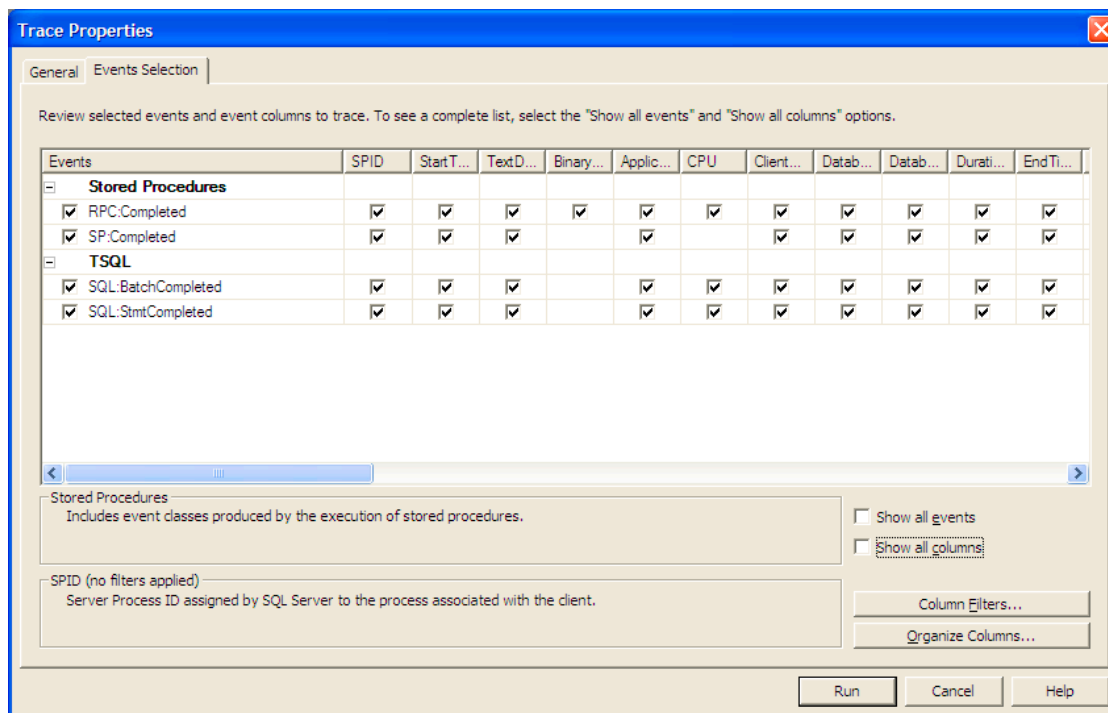
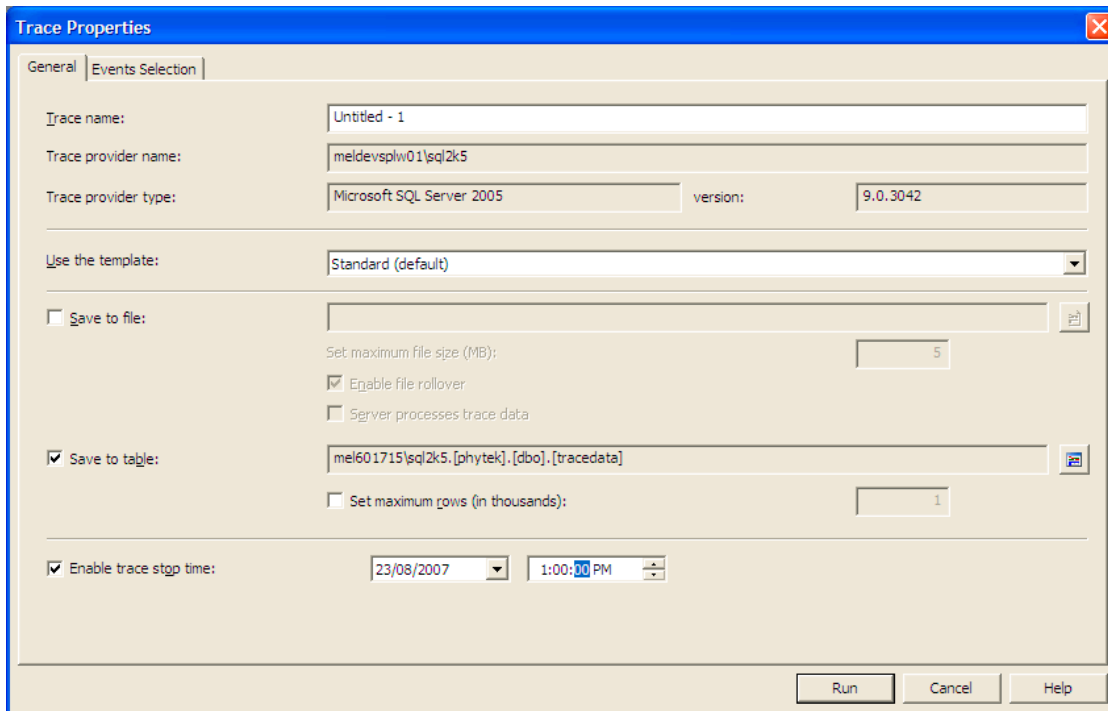


Figure 1.

Figure 2 shows the General tab in the same dialog. I have configured the trace to store into a table on a server other than the server I am tracing. I have also configured the trace to stop after an hour.



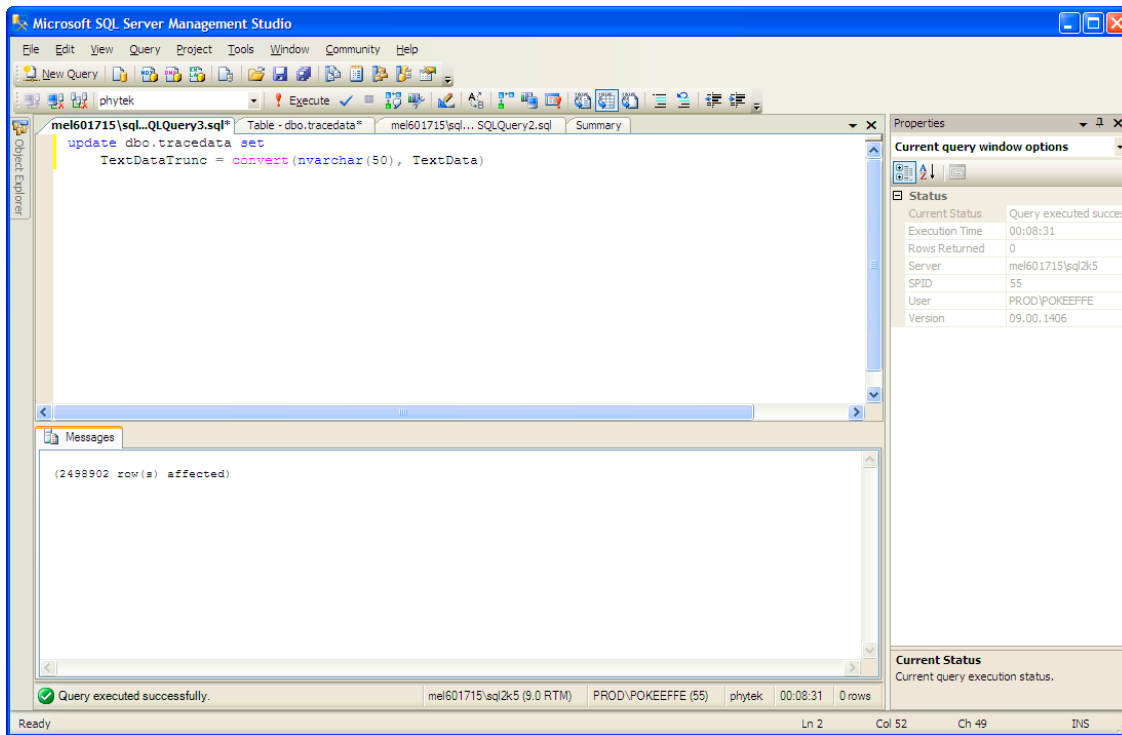
**Figure 2.**

Once the trace is finished, the data should now be available in the database table that I configured. For those who wish to use server side tracing, we will also assume from this point that the trace data now exists in a table.

On a server with a large amount of throughput, there will be a large number of rows in the trace table. In order to make sense of all this data, it will be necessary to aggregate. I suggest aggregating by at least the SQL text, or TextData column. You can include other columns in your aggregation, such as user or client host name, but for now I will concentrate on TextData.

TextData is a text column, which means I can't do a GROUP BY on it. So I will convert it to something we can do a GROUP BY on. In order to do this, I will create a column on the trace table called TextDataTrunc. Figure 3 illustrates the populating of this column with a simple UPDATE.

To get a more accurate aggregation, it would be better to process the TextData column and replace the parameters and literals with some token that allows the SQL statements to be hashed. The hash could then be stored, and the aggregation could be performed on the hash value. This could be done with a C# user-defined function on SQL Server 2005. Illustrating how to do this is beyond the scope of this paper, so I am using the quick and dirty method.



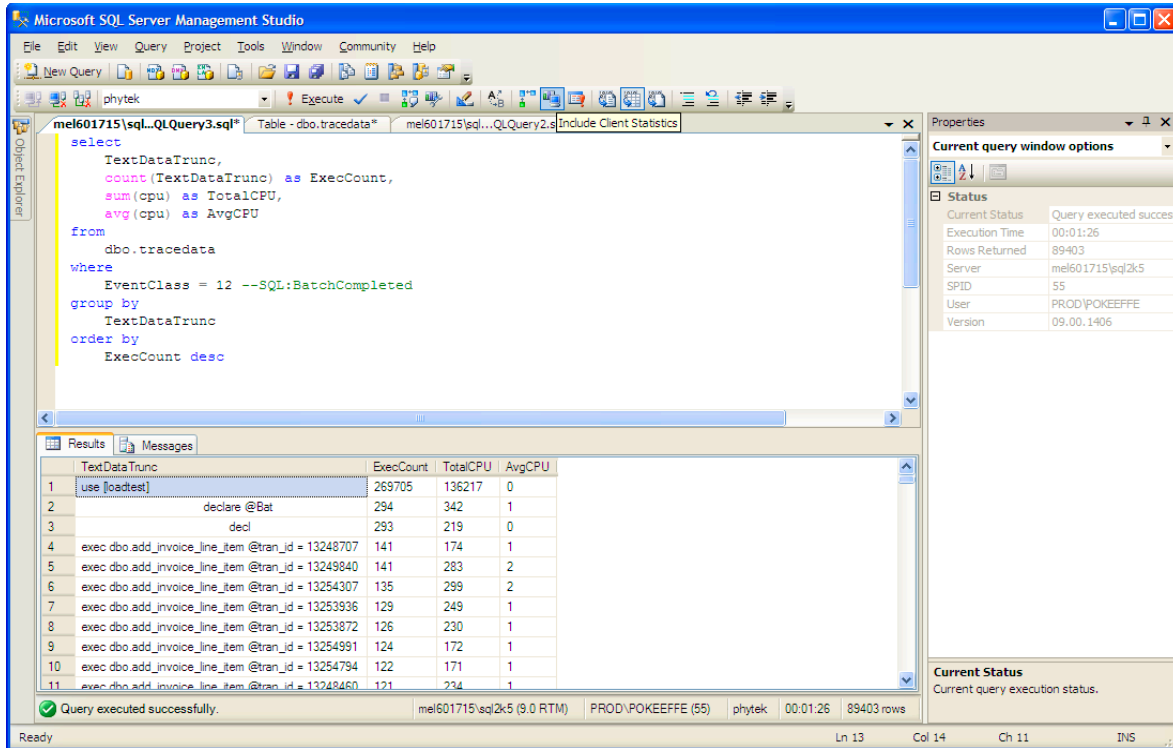
**Figure 3.**

Once the UPDATE is completed, I can now query the table to get the data we require.

For example, say you wanted to know the SQL that had been executed the most—I could use a simple query:

```
select
    TextDataTrunc,
    Count(TextDataTrunc) as ExecCount,
    Sum(cpu) as TotalCPU
    Avg(cpu) as AvgCPU
from
    dbo.tracedata
where
    EventClass = 12
group by
    TextDataTrunc
order by
    ExecCount desc
```

Figure 4 shows an example of this.



**Figure 4**

The values to use for the EventClass column can be found in SQL Server books online under the topic `sp_trace_setevent`.

# 5. Zen and the Art of Negotiating with Your SAN Administrator

---

Storage area networks (SANs) are fantastic. They offer the ability to provision and manage storage in a simple and easy way.

Even though SANs can be configured for fast performance from a SQL Server perspective, they often aren't. Organizations usually implement SANs for reasons such as storage consolidation and ease of management, not for performance. To make matters worse, generally you do not have direct control over how the provisioning is done on a SAN. Thus, you will often find that the SAN has been configured for one logical volume where you have to put all the data files.

Having all the files on a single volume is generally not a good idea if you want the best I/O performance. As an alternative, you will want to:

- Place log files on their own volume, separate from data files. Log files are almost exclusively written and not read. So you would want to configure for fast write performance
- Place tempdb on its own volume. tempdb is used for myriad purposes by SQL Server internally, so having it on its own I/O subsystem will help

To further fine-tune performance, you will first need some stats. There are, of course, the Windows disk counters, which will give you a picture of what Windows thinks is happening (don't forget to adjust raw numbers based on RAID configuration). Also, SAN vendors often have their own performance data available. SQL Server also has file level I/O information available in the form of a function `fn_virtualfilestats`. From this function, you can:

- Derive I/O rates for both reads and writes
- Get I/O throughput
- Get average time per I/O
- Look at I/O wait times

Figure 5 shows the output of a query using this function ordered by `IoStallIMS`, which is the amount of time users had to wait for I/O to complete on a file.



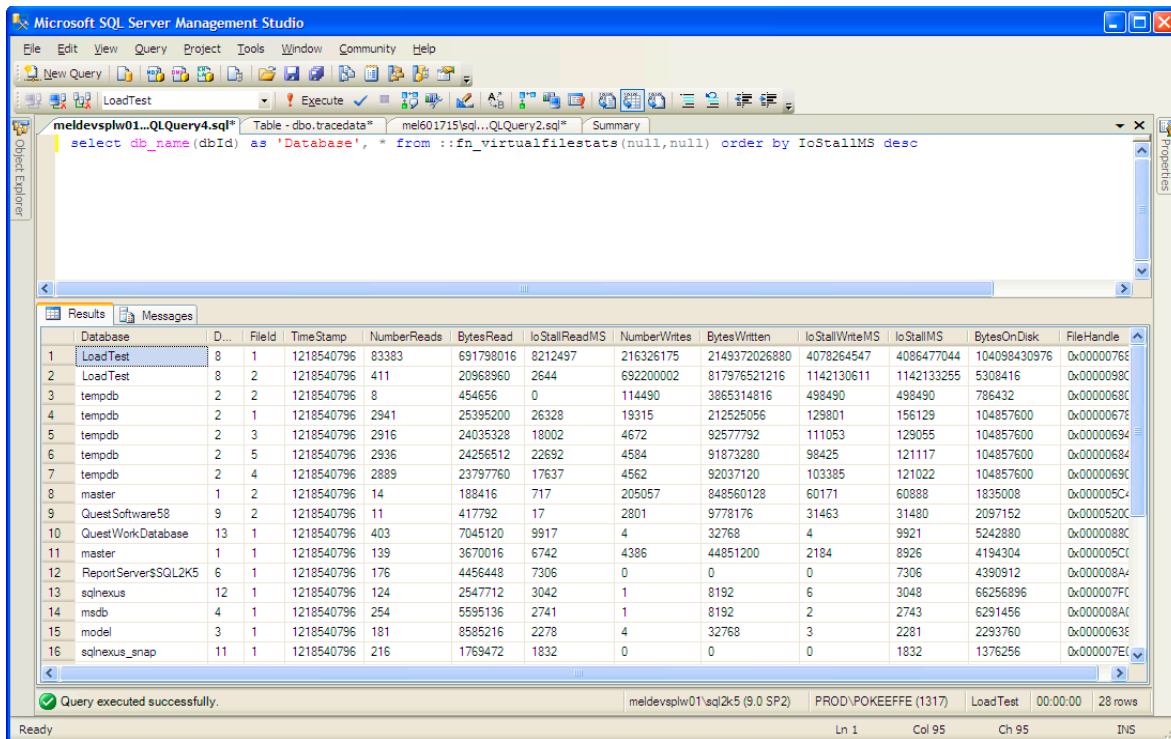


Figure 5.

Using these numbers, you can quickly narrow down which files are responsible for consuming I/O bandwidth and ask questions such as:

- Is this I/O necessary? Am I missing an index?
- Is it one table or index in a file that is responsible?  
Can I put this index or table in another file on another volume

### 3. The Horror of Cursors (and Other Bad T-SQL)

There is a blog I read every day — [www.thedailywtf.com](http://www.thedailywtf.com) (wtf stands for Worse Than Failure, of course). Readers post real experiences they had with bad organizations, processes, people and code. In it I found this gem:

```

DECLARE PatientConfirmRec CURSOR FOR
SELECT ConfirmFlag
FROM Patient where policyGUID = @PolicyGUID
OPEN PatientConfirmRec
FETCH NEXT FROM PatientConfirmRec
WHILE @@FETCH_STATUS = 0
BEGIN
UPDATE Patient
SET ConfirmFlag = 'N'
WHERE CURRENT OF PatientConfirmRec
FETCH NEXT FROM PatientConfirmRec
END
CLOSE PatientConfirmRec
DEALLOCATE PatientConfirmRec

```

This is real code in a real production system. It can actually be reduced to:

```
UPDATE Patient SET ConfirmFlag = 'N'  
WHERE PolicyGUID = @PolicyGUID
```

This refactored code of course will run much more efficiently, allow the optimizer to work its magic and take far less CPU time. In addition, it will be far easier to maintain. It's important to schedule a code review of the T-SQL in your applications, both stored code and client side, and to try to refactor such nonsense.

Bad T-SQL can also appear as inefficient queries that do not use indexes, mostly because the index is incorrect or missing. It's important to learn how to tune queries using query plans in SQL Server Management Studio. Figure 6 shows an example of a large query plan.

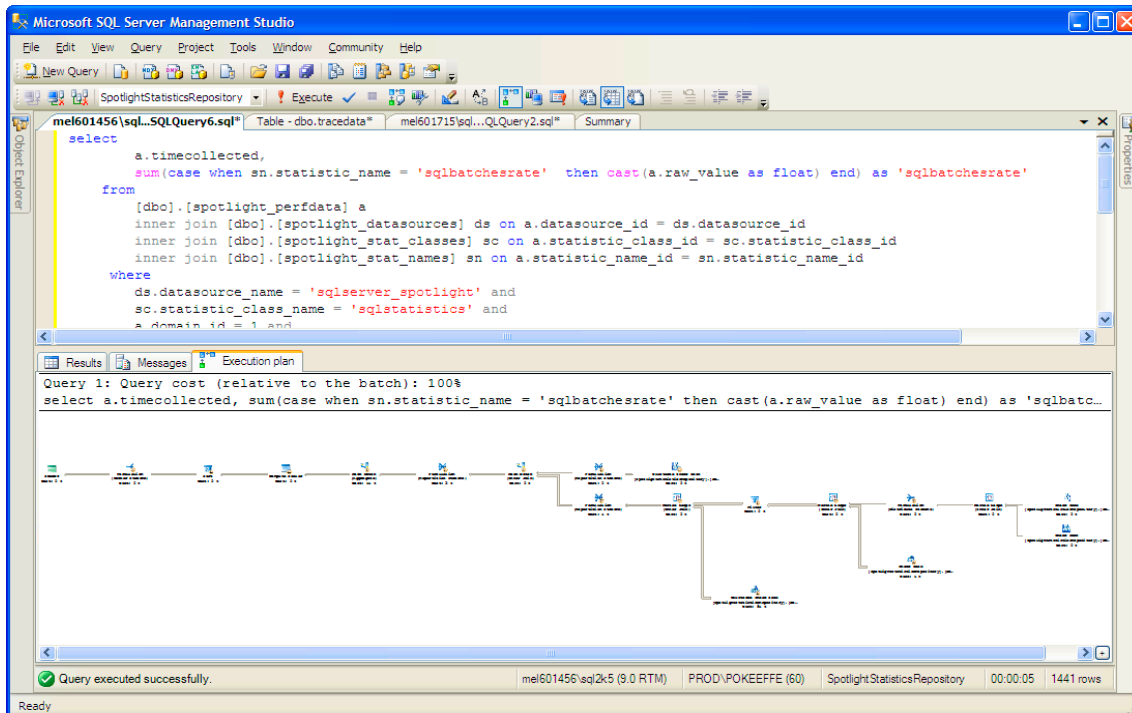


Figure 6

A detailed discussion of query tuning using query plans is beyond the scope of this white paper. However, the simplest way to start this process is by turning SCAN operations into SEEKS. SCANS will read every row in the table. For large tables, it is expensive in terms of I/O, whereas a SEEK will use an index to go straight to the required row. This, of course, requires an index to use, so if you find SCANS in your workload, you could be missing indexes.

There are a number of good books on this topic, including:

- *SQL Server Query Performance Tuning Distilled, Second Edition (paperback)* by Sajal Dam
  - *Microsoft SQL Server 2005 Performance Optimization and Tuning Handbook (paperback)* by Ken England, Gavin JT Powell
4. Plan Reuse – Recycling for SQL

Before executing a SQL statement, SQL Server first creates a query plan. This defines the method SQL Server will use to satisfy the query. Creating a query plan requires significant CPU. Thus, SQL Server will run more efficiently if it can reuse query plans instead of creating a new one each time a SQL statement is executed.

There are some performance counters available in the SQL Statistics performance object that will tell you whether you are getting good plan reuse.

$(\text{Batch Requests/sec} - \text{SQL Compilations/sec}) / \text{Batch Requests/sec}$

This formula tells you the ratio of batches submitted to compilations. You want this number to be as small as possible. A 1:1 ratio means that every batch submitted is being compiled, and there is no plan reuse at all.

It's not easy to pin down the exact workload that is responsible for poor plan reuse, because the problem usually lies in the client application code that is submitting queries.

You therefore may need to look at the client application code that is submitting queries. Is it using prepared parameterized statements?

Using parameterized queries not only improves plan reuse and compilation overhead, but it also reduces the SQL injection attack risk involved with passing parameters via string concatenation.

```
public void ExecuteSomeSQL(int aParam) {
    //cmd is a SqlCommand created somewhere else
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "select foo1, foo2, foo3 from bar where foo1=" + aParam.ToString();

    SqlDataReader dr = cmd.ExecuteReader();
    try {
        while (dr.Read()) {
        }
    } finally {
        dr.Close();
    }
}
```

**Bad**

```
public void ExecuteSomeSQL(int aParam) {
    //cmd is a SqlCommand created somewhere else
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "select foo1, foo2, foo3 from bar where foo1 = @foo1";
    cmd.Parameters["@foo1"].Value = aParam;

    SqlDataReader dr = cmd.ExecuteReader();
    try {
        while (dr.Read()) {
        }
    } finally {
        dr.Close();
    }
}
```

**Good**

**Figure 7**

Figure 7 shows two code examples. Though they are contrived, they illustrate the difference between building a statement through string concatenation and using prepared statements with parameters.

SQL Server cannot reuse the plan from the 'Bad' example. If a parameter had been a string type, this function could be used to mount a SQL injection attack.

The 'Good' example is not susceptible to a SQL injection attack because a parameter is used, and SQL Server is able to reuse the plan.

## 2. The Mystery of the Buffer Cache

---

The buffer cache is a large area of memory used by SQL Server to optimize physical I/O.

No SQL Server query execution reads data directly off the disk. The database pages are read from the buffer cache. If the sought-after page is not in the buffer cache, a physical I/O request is queued. Then the query waits and the page is fetched from the disk.

Changes made to data on a page from a DELETE or an UPDATE operation are also made to pages in the buffer cache. These changes are later flushed out to the disk.

This whole mechanism allows SQL Server to optimize physical I/O in several ways:

- Multiple pages can be read and written in one I/O operation
- Read ahead can be implemented. SQL Server may notice that for certain types of operations, it could be useful to read sequential pages—the assumption being that right after you read the page requested, you will want to read the adjacent page

There are two indicators of buffer cache health:

1. `MSSQL$Instance:Buffer Manager\Buffer cache hit ratio` – This is the ratio of pages found in cache to pages not found in cache. Thus, the pages need to be read off disk. Ideally, you want this number to be as high as possible. It is possible to have a high hit ratio but still experience cache thrashing.
2. `MSSQL$Instance:Buffer Manager\Page Life Expectancy` – This is the amount of time that SQL Server is keeping pages in the buffer cache before they are evicted. Microsoft says that a page life expectancy greater than five minutes is fine. If the life expectancy falls below this, it can be an indicator of memory pressure (not enough memory) or cache thrashing.

Cache thrashing is the term used when a large table or index scan is occurring. Every page in the scan must pass through the buffer cache. This is very inefficient because the cache is being used to hold pages that are not likely to be read again before they are evicted.

Since every page must pass through the cache, other pages need to be evicted to make room. A physical I/O cost is incurred because the page must be read off disk. Cache thrashing is usually an indication that large tables or indexes are being scanned.

To find out which tables and indexes are taking up the most space in the buffer cache, you can examine the `cacheobjects` on SQL Server 2000 or `sys.dm_os_buffer_descriptors` on SQL Server 2005.

The example query below illustrates how to access the list of tables/indexes that are consuming space in the buffer cache on SQL Server 2005:

```

select
    o.name,
    i.name,
    bd.*
from
    sys.dm_os_buffer_descriptors bd
    inner join sys.allocation_units a on bd.allocation_unit_id =
a.allocation_unit_id
    inner join sys.partitions p      on (a.container_id = p.hobt_id and a.type
in (1,3)) or (a.container_id = p.partition_id and a.type = 2 )
    inner join sys.objects o on p.object_id = o.object_id
    inner join sys.indexes i on p.object_id = i.object_id and p.index_id =
i.index_id

```

You can also use the new index DMVs to find out which tables/indexes have large amounts of physical I/O.

# 1. The Tao of Indexes

SQL Server 2005 gives us some very useful new data on indexes.

## sys.dm\_db\_index\_operational\_stats

sys.dm\_db\_index\_operational\_stats contains information on current low-level I/O, locking, latching and access method activity for each index.

Use this DMV to answer the following questions:

- Do I have a 'hot' index? Do I have an index on which there is contention? – The row\_lock\_wait\_in\_ms/page\_lock\_wait\_in\_ms columns can tell us whether there have been waits on this index
- Do I have an index that is being used inefficiently? Which indexes are currently I/O bottlenecks? – The page\_io\_latch\_wait\_ms column can tell us whether there have been I/O waits while bringing index pages into the buffer cache – a good indicator that there is a scan access pattern
- What sort of access patterns are in use? The range\_scan\_count and singleton\_lookup\_count columns can tell us what sort of access patterns are used on a particular index

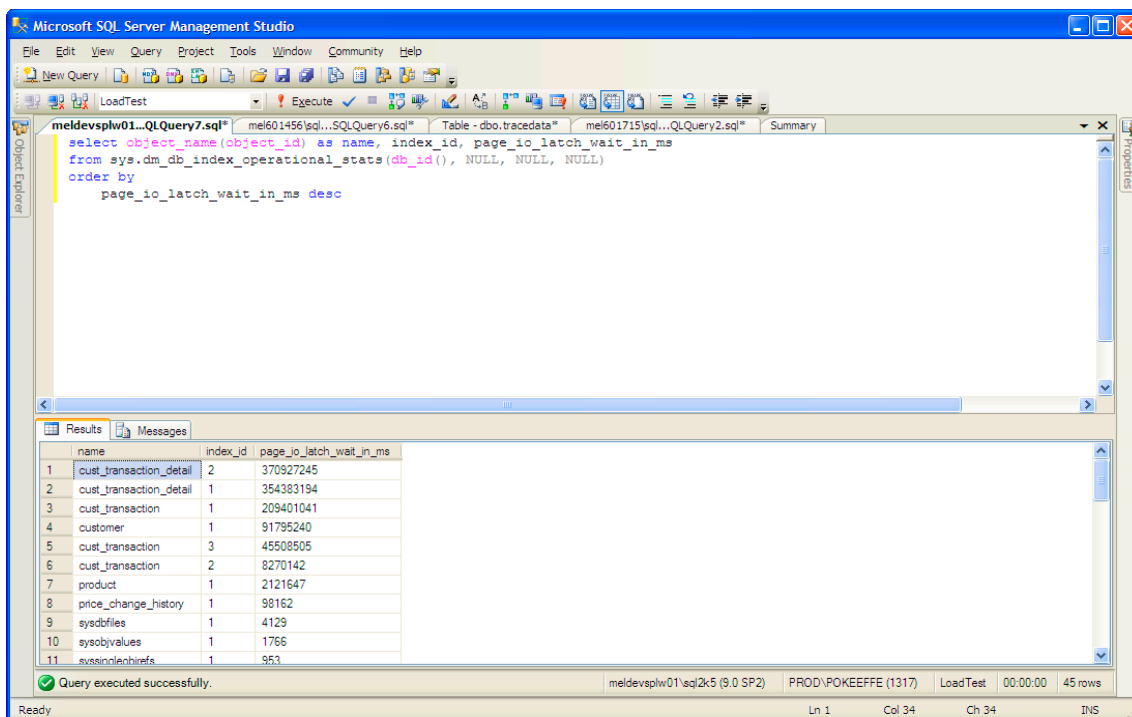


Figure 8.

Figure 8 illustrates the output of a query that lists indexes by the total PAGE\_IO\_LATCH wait. This is very useful when trying to determine which indexes are involved in I/O bottlenecks.

## sys.dm\_db\_index\_usage\_stats

sys.dm\_db\_index\_usage\_stats contains counts of different types of index operations and the time each type of operation was last performed.

Use this DMV to answer the following questions:

- How are users using the indexes? The user\_seeks, user\_scans, user\_lookups columns can tell you the types and significance of user operations against indexes
- What is the cost of an index? The user\_updates column can tell you what the level of maintenance is for an index
- When was an index last used? The last\_\* columns can tell you the last time an operation occurred on an index

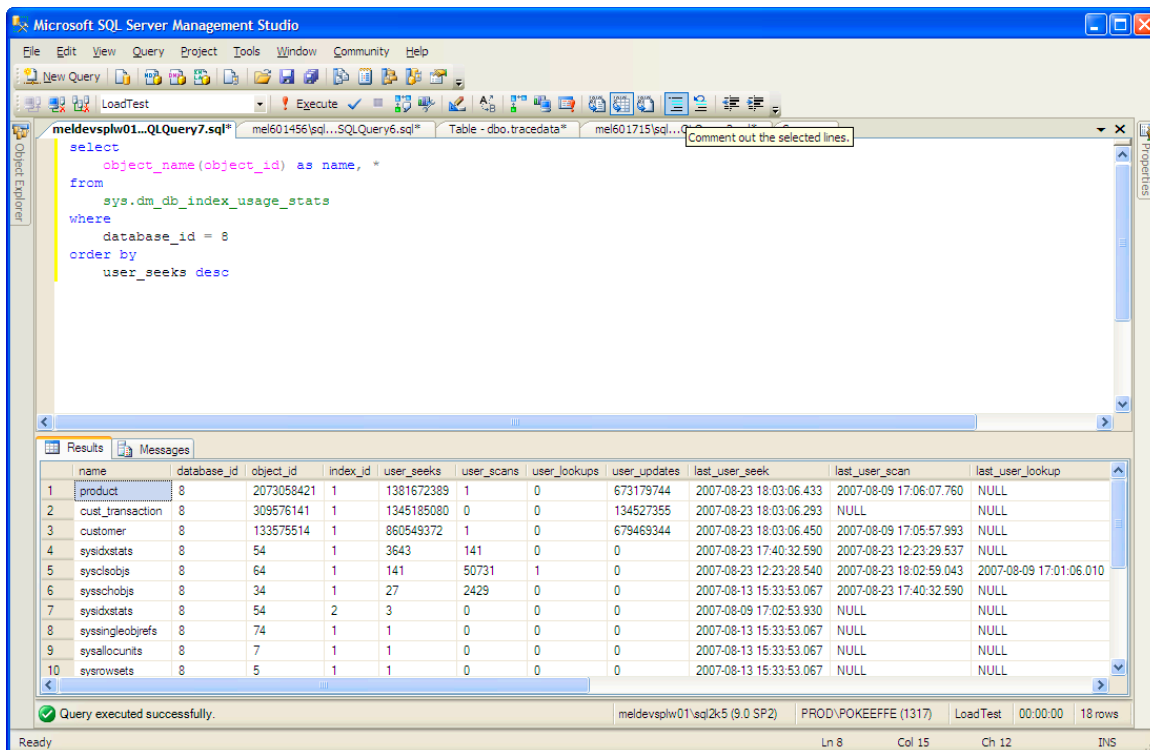


Figure 9.

Figure 9 illustrates the output of a query that lists indexes by the total number of user\_seeks. If you instead wanted to identify indexes that had a high proportion of scans, you could order by the user\_scans column.

Now that you have an index name, wouldn't it be good if you could find out what SQL statements used that index?

On SQL Server 2005 – now you can.

# And an Extra One for Good Measure. Learn XPath

Say we had an index name received from a query on `sys.dm_os_buffer_descriptors` or from `sys.dm_db_index_operational_stats`. How do we find the SQL Statements that use that index?

Microsoft did two really cool things (amongst all the other really cool things) in SQL Server 2005:

1. Exposed the contents of the procedure (or plan) cache through `sys.dm_exec_query_stats` and more specifically the column containing the compiled plan in XML
2. Embedded the ability to use XPath expressions on XML columns

XPath is a way of querying an XML DOM (Document Object Model). What are the SQL Statements that use a particular index? An example is shown below.

```
select sql.text,  
       substring(sql.text, statement_start_offset/2,  
                (case when statement_end_offset = -1 then  
                    len(convert(nvarchar(max), text)) * 2  
                    else statement_end_offset end - statement_start_offset)/2),  
       qs.execution_count,  
       qs.*,  
       p.*  
from  
     sys.dm_exec_query_stats as qs  
     cross apply sys.dm_exec_sql_text(sql_handle) sql  
     cross apply sys.dm_exec_query_plan(plan_handle) p  
where  
     query_plan.exist('declare default element namespace  
"http://schemas.microsoft.com/sqlserver/2004/07/showplan";  
/ShowPlanXML/BatchSequence/Batch/Statements//Object/@Index[. =  
"[IDX_cust_transaction_cust_id]"']) = 1
```



Here is another example of all the SQL Statements that do a clustered index scan.

```
declare @op nvarchar(30)
set @op = 'Clustered Index Scan'

select top 10
    sql.text,
    substring(sql.text, statement_start_offset/2, (case when statement_end_offset
= -1 then len(convert(nvarchar(max), text)) * 2 else statement_end_offset -
statement_start_offset)/2),
    qs.execution_count, qs.*, p.*
from sys.dm_exec_query_stats AS qs
cross apply sys.dm_exec_sql_text(sql_handle) sql
cross apply sys.dm_exec_query_plan(plan_handle) p
where query_plan.exist('
declare default element namespace
"http://schemas.microsoft.com/sqlserver/2004/07/showplan";
/ShowPlanXML/BatchSequence/Batch/Statements//RelOp/@PhysicalOp[. =
sql:variable("@op")]
') = 1 and total_logical_reads/execution_count > 1000
order by total_logical_reads
```

There are many other ways you can use this mechanism.

For example, SQL Server 2005 plans have data that tells you that SQL Server would have used an index on a column had it been present. You can use this method to find these plans.

You can find queries that have large estimates for the numbers of rows being read to identify queries that read large numbers of rows.

You can identify lookup operations that could be optimized using a covering index. Since the columns to be returned are known, and the indexes that the plan used are known, you can add columns to the index so it can be used as a covering index.

# Conclusion

---

On reflection, there are far more than ten (or eleven) things you should know about SQL Server performance. However, this white paper offers a good starting point and some practical tips about performance optimization that you can apply to your SQL Server environment.

# About the Author

---

**Patrick O'Keeffe** is a senior software architect at Quest Software, where he specializes in the design and implementation of diagnostic and performance tools for Microsoft SQL Server. Patrick has more than 12 years of experience in software engineering and architecture, and is based in Quest's Melbourne Office.

# Notes

---

<sup>1</sup> When using Windows disk counters, it is necessary to adjust for RAID configurations. You can do this using the following formulas:

Raid 0	value = (reads + writes) / number of disks
Raid 1	value = [reads + (2 * writes)] / 2
Raid 5	value = [reads + (4 * writes)] / number of disks
Raid 10	value = [reads + (2 * writes)] / number of disks

# Quest's Performance Management Solutions for Microsoft SQL Server

---

Written by  
Quest Software, Inc.

© 2009 Quest Software, Inc.  
**ALL RIGHTS RESERVED.**

This document contains proprietary information protected by copyright. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose without the written permission of Quest Software, Inc. (“Quest”).

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST’S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters  
LEGAL Dept  
5 Polaris Way  
Aliso Viejo, CA 92656  
**www.quest.com**  
email: **legal@quest.com**

Refer to our Web site for regional and international office information.

## Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, ChangeManager, Defender, DeployDirector, Desktop Authority, DirectoryAnalyzer, DirectoryTroubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, IWatch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, MultSess, NBSPool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point,Click,Done!, PowerGUI, Quest Central, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, Security Lifecycle Map, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vControl, vConverter, vFoglight, vOptimizer, vRanger, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vBackup, Vizioncore vEssentials, Vizioncore vMigrator, Vizioncore vReplicator, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

Updated— May 2009

# Contents

---

- Quest's SQL Server Performance Management Tools .....96
  - Foglight® Performance Analysis for SQL Server .....96
    - Overview .....96
    - Product Architecture .....97
    - Product Features and Displays .....97
  - Spotlight® on SQL Server Enterprise .....103
    - Overview .....103
    - Product Architecture .....103
    - Product Features and Displays .....104
- Understanding and Resolving Common Performance Bottlenecks .....109
  - CPU Bottlenecks .....109
    - CPU Contention from Other Applications on the SQL Server .....109
    - Inefficient Query Plans .....109
    - Excessive or Redundant Recompiles .....110
    - Hardware and SQL Server Configuration Issues .....112
    - Troubleshooting CPU Contention Using Quest Tools .....112
  - I/O Bottlenecks .....119
    - Using Performance Counters .....119
    - Wait Event Analysis .....120
    - Troubleshooting I/O Bottlenecks Using Quest Tools .....121
  - Memory Pressure .....128
    - Types of Memory .....128
    - Relationship between O/S Architecture and SQL Server Architecture .....128
    - Types of Memory Pressure .....129
    - Troubleshooting Memory Pressure Using Quest Tools .....130
  - TempDB Troubleshooting .....138
    - Objects .....138
    - Optimizations .....138
    - Configuration and Sizing .....139
    - Troubleshooting TempDB Using Quest Tools .....141
- Recommended Reading .....144

# Quest's SQL Server Performance Management Tools

---

Quest offers two valuable tools for SQL Server performance management: Foglight® Performance Analysis for SQL Server and Spotlight® on SQL Server Enterprise. This technology overview provides a solid introduction to both tools, including a detailed description of their features and displays. Then the paper explains several key performance issues—CPU bottlenecks, I/O bottlenecks, memory pressure, and TempDB issues—and explains how Performance Analysis and Spotlight can help you quickly identify and resolve these issues.

## Foglight® Performance Analysis for SQL Server

### Overview

Business success is tied to the productivity of enterprise databases, which are constantly changing. Therefore, IT staffers must be able to proactively diagnose and resolve bottlenecks and scalability issues that threaten performance. However, effective SQL workload performance tuning and troubleshooting is an enormous task that can consume a great deal of a DBA's time. Tasks include the following:

- Establishing benchmarks to understand the load and throughput the database systems can handle
- Collecting metrics and monitoring activity around the clock
- Performing both real-time and historical analyses to detect issues and determine proper action plans

With the proper tools to help with these tasks, DBAs can maintain operational integrity and end-user satisfaction, while avoiding costly production slowdowns. Foglight Performance Analysis for SQL Server enables DBAs to quickly and efficiently determine the root cause of performance deviations, perform comprehensive analyses, and resolve performance bottlenecks.



## Product Architecture

Performance Analysis is built upon the StealthCollect® agent framework: a multi-tiered, memory sampling technology that captures operating system and SQL Server instance and workload data up to 50 times per second (figure 1). Because Windows memory is read, no direct connections to SQL Server are necessary, which makes this technology the lowest impact complete collection solution available.

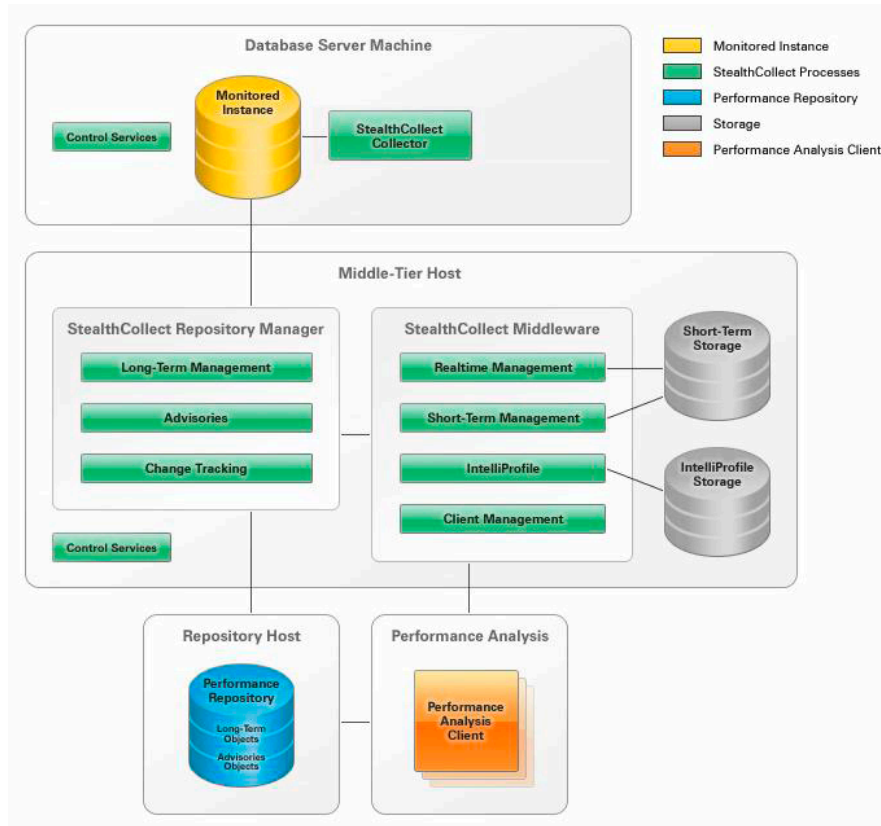
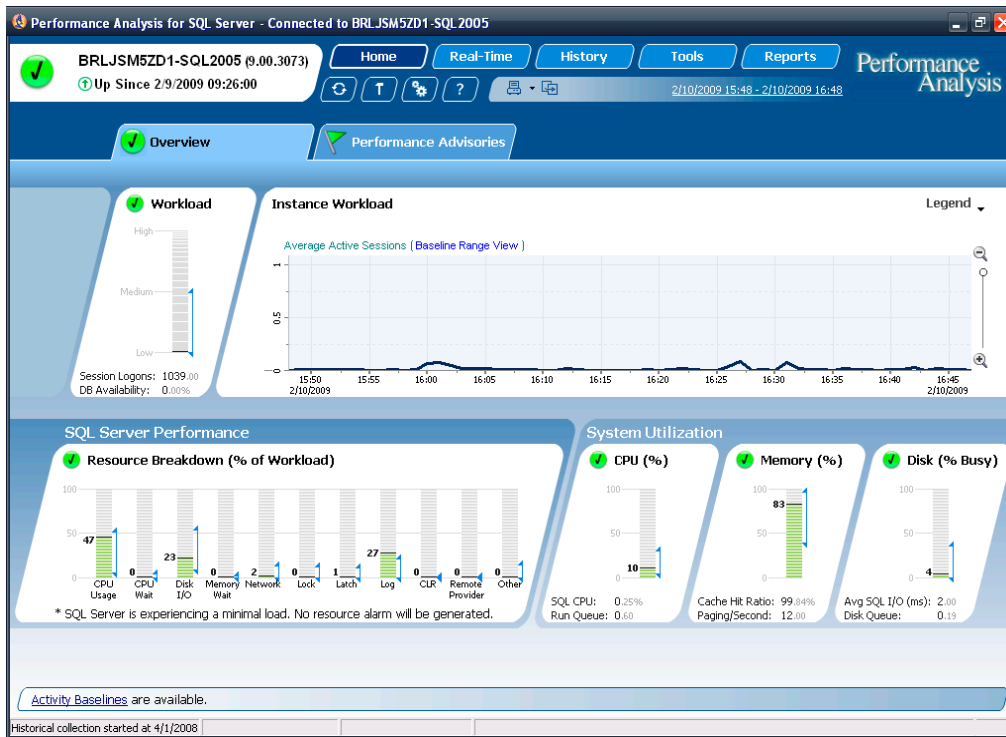


Figure 1—StealthCollect framework

The middleware component is responsible for all data aggregation, averaging and storage, and also implements the IntelliProfile® advanced learning baseline technology to baseline all instance-level data collected. The repository database provides change tracking and performance advisories. Performance advisories act as an in-house SQL Server performance expert—always analyzing performance data and providing specific advice to correct performance anomalies in the context they are observed.

## Product Features and Displays

**The Home Page Dashboard** (figure 2) displays between five minutes and 24 hours of activity, based on the setting you choose using the time selector in the upper right of the screen. The dashboard presents information on the instance workload and on the amount of each resource group consumed by both the SQL Server workload (bottom left corner under “SQL Server Performance”) and by Windows overall (lower right corner under “System Utilization”). Baseline indicators show how the activity for each resource category compares to what is normal for the time range displayed.



**Figure2—Foglight Performance Analysis for SQL Server Homepage dashboard**

Gauges are colored green, yellow, orange or red, indicating increasing levels of severity. You can click a gauge to be informed of resource deviations relative to either instance-level thresholds or the baseline operating range. You can then click through directly to the History View to continue your investigation with a focus on that resource's activity. Flag icons next to each gauge highlight performance advisories that pertain specifically to each resource.

**The Real-Time View** (figure 3) displays how the SQL Server workload (top timeline graph) correlates with individual metrics, which are visible in a tabular layout in the bottom third of the screen for up to the last hour of activity. Clicking a particular metric displays a graph of its values over the time range specified. Mousing over the bar to the right of a metric displays a popup tool tip showing a component breakdown. The Top Sessions tab displays the sessions consuming the most resources over the time range selected. Selecting Current Sessions in the left panel displays current session activity similar to information displayed by running `sp_who2` via native tools.

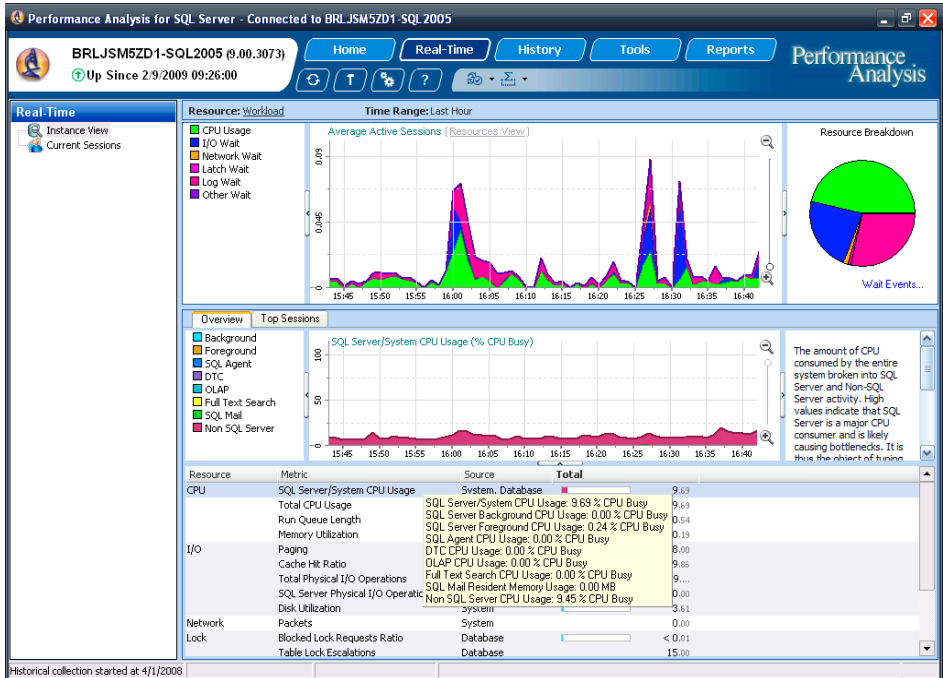


Figure 3—Foglight Performance Analysis for SQL Server workload view

The **History View** displays any advisories raised during the timeframe, an overview of metric performance as in the instance view, a time breakdown, and a list of changes detected during the selected time range (figure 4). As with any Performance Analysis display, changing the resource from Workload (all activity) to a specific resource category focuses your investigation on only aspects of your workload that generated that resource category; any drill-down activity in the left panel will display only information relative to that resource category.

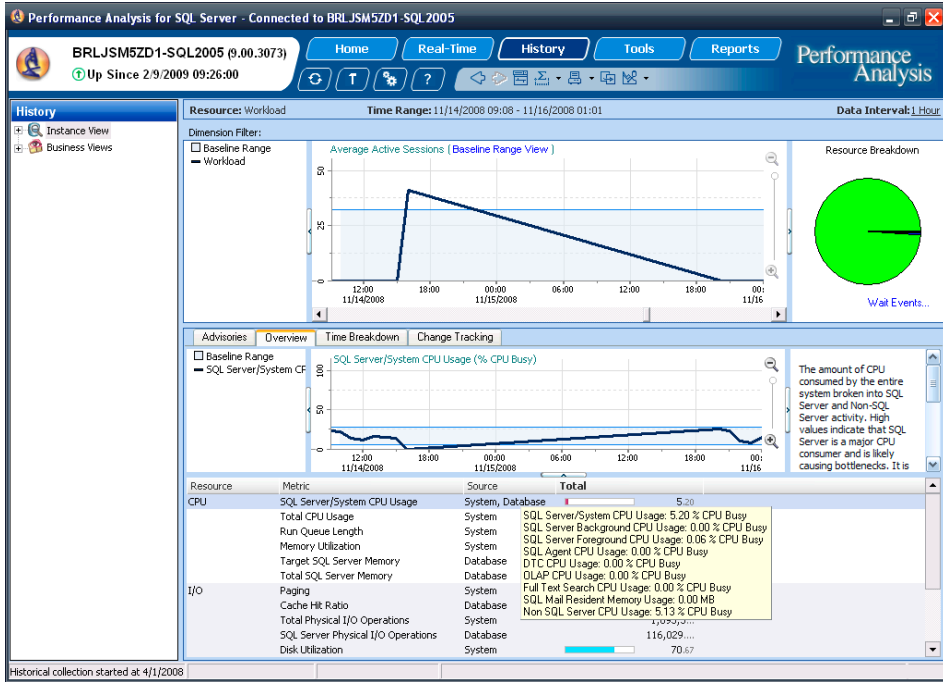
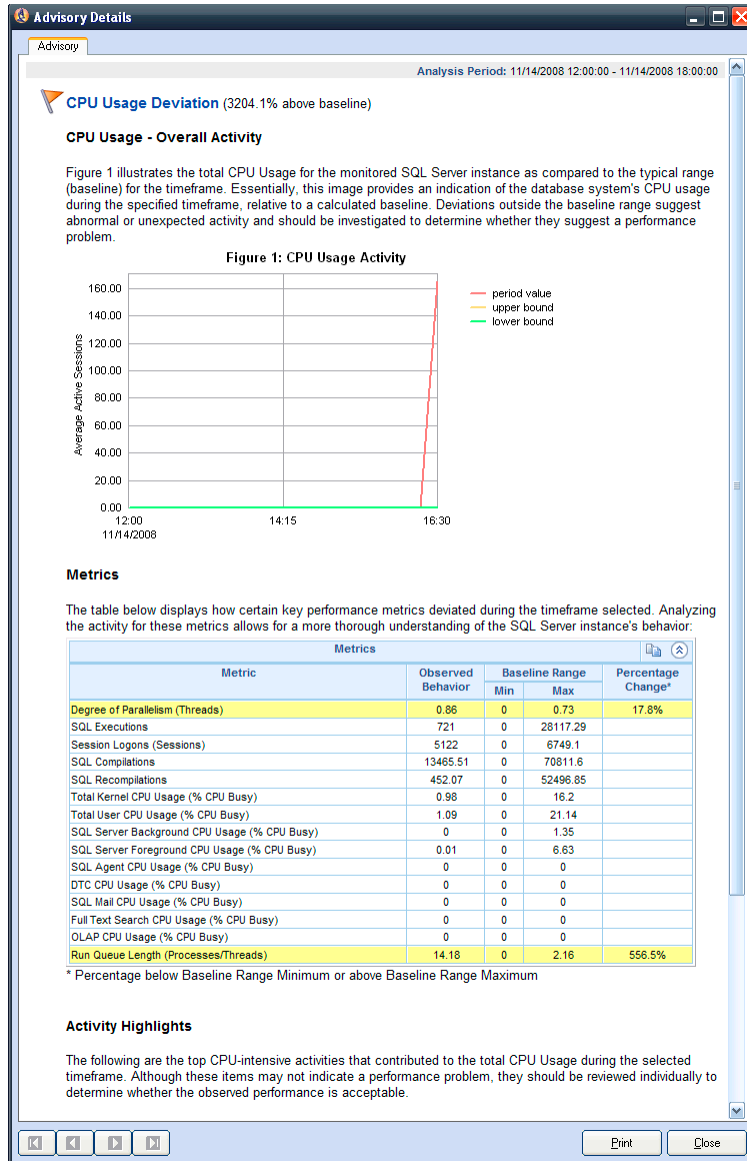


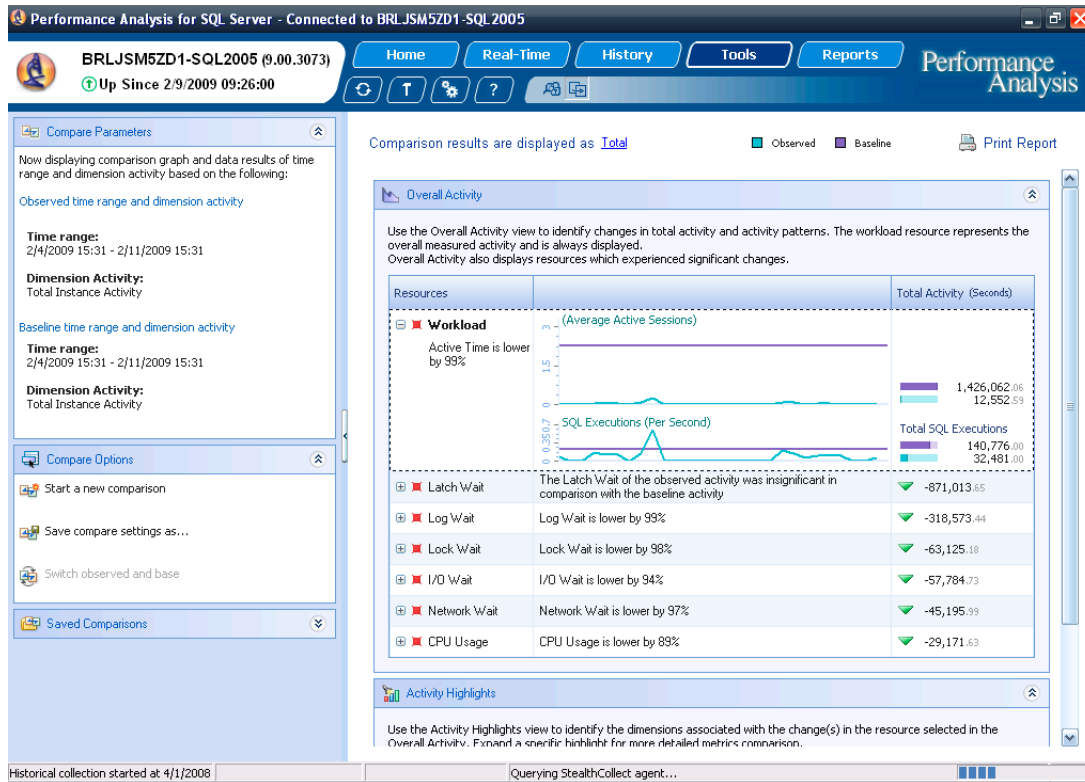
Figure 4—Foglight Performance Analysis for SQL Server History View

**Baseline Deviation and Performance Advisories** display resource and time-sensitive advice to quickly identify the root cause of performance anomalies and recommendations for optimizing application workload performance. Advisories (figure 5) are presented either in the context of a related resource category (such as the flag icons on the home page) or as part of a complete resource-maximizing action plan. All advisory data is available historically, so as you troubleshoot performance anomalies and workload trends over time, you'll have the benefit of the advisory data to guide your decision-making and investigation processes. Each advisory provides rationale, recommended actions, real-time metrics and workload observations, background information to further explain the condition, and links to Performance Analysis history so you can immediately launch an investigation.



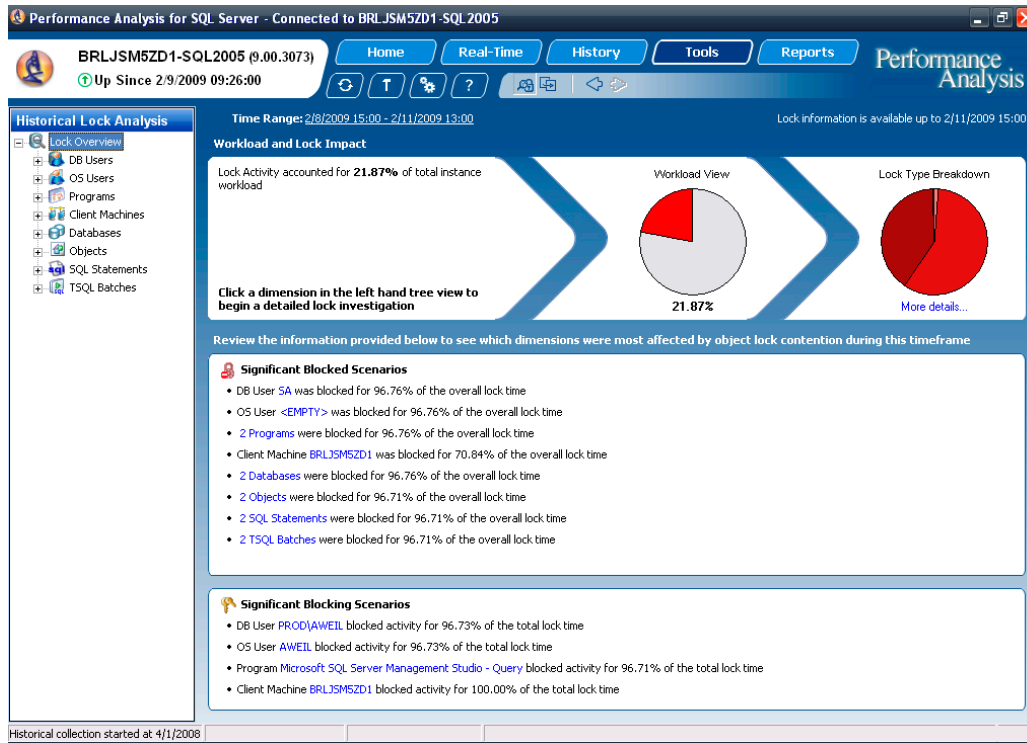
**Figure 5—Foglight Performance Analysis for SQL Server Performance Advisory**

The **Compare Utility** lets you answer the question, “How can I tell what’s different?” You can access this utility from the Tools menu option along the top of the console (figure 6), or by right-clicking anywhere in the History tree view. Comparisons can be made between observed instance activity and the baseline, between observed activity between timeframes, or between any dimension activity (SQL Statements, Programs, etc.) and over varying timeframes. Comparison definitions can be saved and executed on a schedule for automated email or file-based report delivery.



**Figure 6—Foglight Performance Analysis for SQL Server Compare Utility**

**Historical Lock Analysis** provides detailed evaluations of historical blocking lock scenarios. You can access Historical Lock Analysis from the Tools menu option along the top of the console, or by right-clicking in a session or History tree view and selecting the Historical Lock Analysis option. The interface (figure 7) provides information on the impact of blocking locks on your workload, significant blocking or blocked dimensions, and detailed database and object-level scenarios that highlight the statements that waited and the possible blocking SQL by scenario.



**Figure 7—Foglight Performance Analysis for SQL Server historical lock analysis**

**Reports** are provided on a wide variety of topics, including Activity, Throughput, Tops, Trends, Time Breakdowns, Change Tracking and Compare; Performance Analysis also provides an Executive Workload Summary report for key decision makers. All reports can be customized to suit your needs, and you can create new reports quickly and easily using the report definition wizard for each report category. Reports can be created in a variety of formats, including HTML, PDF (figure 8), RTF and Excel, and you can schedule any report to be delivered via email and/or written to a location of your choosing. Scheduled reports are implemented as Windows tasks, so there is no need to dedicate a machine to run the Performance Analysis console to generate your reports.

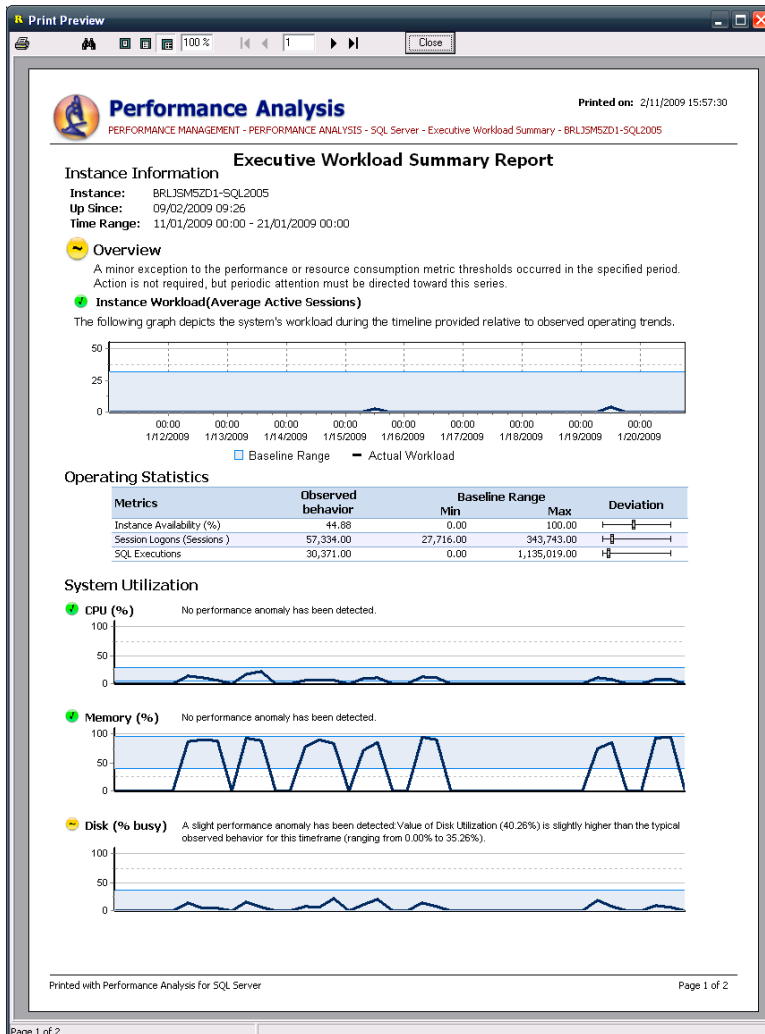


Figure 8—Foglight Performance Analysis for SQL Server executive workload summary report

## Spotlight® on SQL Server Enterprise

### Overview

To ensure business continuity, DBAs need an operational performance view of their SQL Server environments. Spotlight on SQL Server Enterprise provides an overview of enterprise performance at a glance, detecting and diagnosing performance issues across your SQL Server environment. You are instantly alerted to performance bottlenecks detected at the server level, and metric-sensitive alarms point you to the root causes of performance issues. The Xpert tuning module resolves SQL performance issues by ensuring optimal code quality and providing indexing options.

### Product Architecture

Spotlight on SQL Server Enterprise provides 24x7 data collection for your SQL Server instance environment and the Windows servers in your enterprise remotely (figure 9). Because Spotlight's collection methodology is based on stored procedures, the impact of collecting a complete set of diagnostic data on your SQL environment is significantly lower than other ad-hoc SQL-based methods. The Playback database provides up to seven days of recent performance data to allow you to investigate performance anomalies using the same workflow you would employ in a real-time diagnosis and analysis scenario. The Spotlight Statistics Repository adds the ability to perform long-term trending and problem determination.

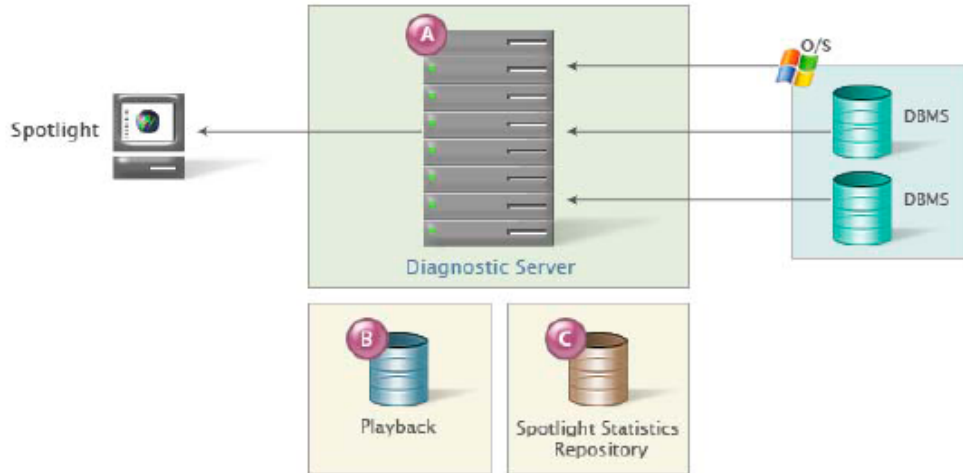


Figure 9—Spotlight on SQL Server Enterprise architecture

## Product Features and Displays

**Spotlight Today** is your landing page for Spotlight on SQL Server Enterprise (figure 10). This consolidated view of your enterprise provides the most recent alarms for your instances and Windows servers, along with helpful links to begin your diagnostic and troubleshooting initiatives. Whether you want to quickly identify your most expensive SQL, manage alarms, view historical data, or configure custom counters Spotlight Today provides a jumpstart from a single location.

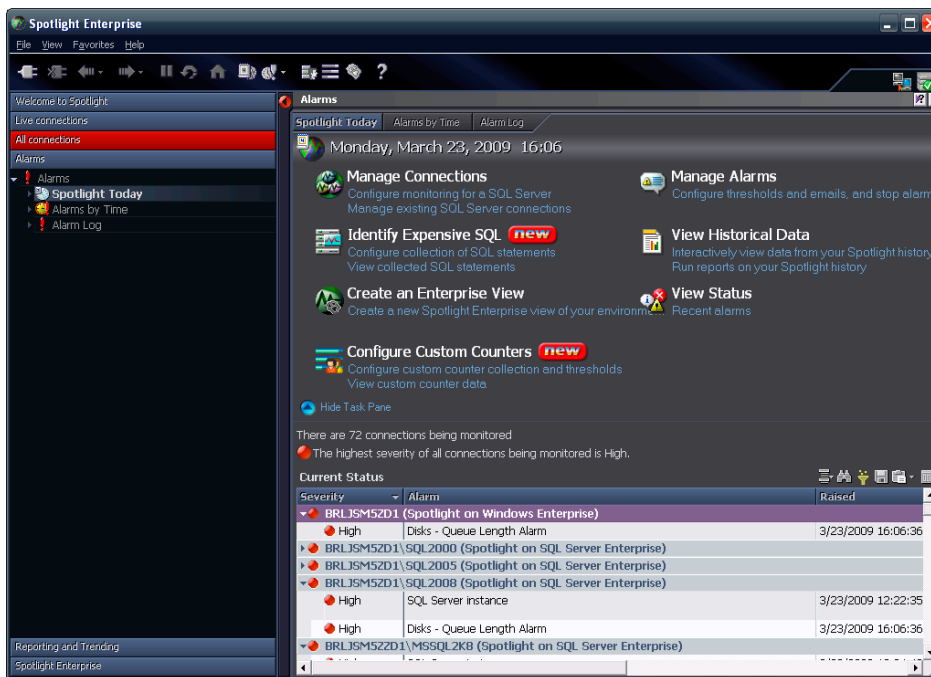


Figure 10—Spotlight Today landing page



The **SQL Server Homepage Dashboard** is Spotlight's proprietary graphical representation of SQL Server's architecture (figure 11). Icons are color-coded to indicate severity, so you can easily determine which area of SQL Server is experiencing a slowdown. Clicking on any yellow or red icon displays helpful alarm text and a link to detailed resource diagnostics. The playback slider control, available from any Spotlight diagnostic dashboard, takes you immediately to a point in time up to seven days in the past, enabling you to diagnose the health of your SQL Server instance using the same interface you use for real-time analysis.



**Figure 11—Spotlight on SQL Server Enterprise home page**

**SQL Server Drill-down Dashboards** present all the diagnostic data you need to determine the root cause of a performance anomaly (figure 12). Whether you navigate from an alarm on the Home Page or manually traverse the drill-down views, you can see the status of your SQL Server instance architecture and understand your workload trends and wait event consumption. You can create custom counters to define any business-related metric or other useful information not included in Spotlight's default collection set. The SQL Analysis feature allows you to leverage either SQL Server rowset or server-side tracing, including an option to set up a scheduled server-side poll of your SQL activity to define the most expensive SQL in your environment.



Figure 12—Spotlight on SQL Server Enterprise SQL Summary drilldown

**Reporting and Trending Views** help you get the big picture on your entire monitoring environment by displaying trending information on a myriad of configuration and performance metrics and statistics (figure 13). You can see how your infrastructure is configured, how your databases have been growing, and the way your workload processing has developed over time. You can also gather information on the custom counters you've defined across your monitoring infrastructure.



Figure 13—Spotlight on SQL Server Enterprise Custom Counters

**The Windows Homepage Dashboard** is Spotlight's proprietary graphical representation of the Windows architecture (figure 14). Icons are color-coded to indicate severity, so you can easily determine which area of your Windows server is experiencing a slowdown. Clicking on any yellow or red icon displays helpful alarm text and a link to detailed resource diagnostics. The playback slider control, available from any Spotlight diagnostic dashboard, takes you to a point in time up to seven days in the past, enabling you to diagnose the health of your SQL Server instance using the same interface you would use for real-time analysis.



**Figure 14—Spotlight's Windows home page dashboard**

**Windows Drill-down Dashboards** present all the diagnostic data you need to determine the root cause of a performance anomaly. Whether you navigate from an alarm on the Home Page or manually traverse the drill-down views, you can see the status of your Windows server architecture and understand your application processing trends. You can create custom counters to define any business-related metric or other useful information not included in Spotlight's default collection set (figure 15).



Figure 15—Spotlight's Windows drilldown into disk activity

The **Topology (Enterprise) View** is Spotlight's graphical representation of your monitored infrastructure (figure 16). Because this view is not tied to the physical deployment of your SQL Server instances or Windows servers, you have the flexibility to define views that make sense to you and your team. Each node in the display identifies whether a problem exists with a given instance or the Windows server it's running on, and pop-up notifications identify the critical alarms that have been raised for each connection.



Figure 16—Spotlight on SQL Server Enterprise topological view

# Understanding and Resolving Common Performance Bottlenecks

---

Major types of performance bottlenecks DBAs must identify and resolve include the following:

- CPU bottlenecks
- I/O bottlenecks
- Memory pressure
- TempDB issues

Let's look at each in turn, and see how Foglight Performance Analysis for SQL Server and Spotlight on SQL Server Enterprise can help you diagnose and resolve these performance issues.

## CPU Bottlenecks

Performance problems related to CPU can be caused by a number of factors, including the following:

- CPU contention from other applications on the SQL Server
- Inefficient query plans
- Redundant recompiles
- Poor parallel query performance
- Hardware SQL Server configuration issues

While this is by no means a complete list, troubleshooting CPU bottlenecks in the order specified above can be a good action plan. In particular, addressing server CPU (a configuration issue) can be a necessary step, it should be a last resort. Adding additional CPU can alleviate some CPU-related issues on a SQL Server, but if the problem is based on the application workload, the problem may simply re-emerge later. Most decision-makers in an organization will be nonplussed to hear that the CPU purchase they approved was unnecessary because a configuration or application tuning task was actually in order.

### CPU Contention from Other Applications on the SQL Server

Many organizations heed the long-standing Microsoft best practice of minimizing the number of applications running on a SQL Server host machine to ensure optimal performance. If adhering to this practice is not possible, one of the first steps in determining the root of a CPU-related slowdown should be to evaluate the non-SQL Server processes running on the machine.

### Inefficient Query Plans

CPU-related performance problems can also be caused by inefficient query plans. When the SQL Server optimizer generates an execution plan for a query, it attempts to find the fastest means of returning the data requested while balancing the query's impact on I/O and CPU resources—that is, attempting to ensure that neither resource becomes overtaxed. However, the optimizer sometimes makes poor choices, and the selected query plan might tax system resources and result in bottlenecks. For example, queries that seem to effectively utilize indexes defined on the target tables can become I/O bound if too much data is accessed at one time. Queries that rely on Hash and Sort operators are usually not I/O-bound; however, the amount of scanning performed (pre-fetching data so it will be available in the buffer cache) can tax a system's CPU resources and generate overall slowdowns within the instance as other sessions compete for processor time.

SQL Server's Query optimizer uses two main criteria when evaluating the possible paths to take when retrieving data requests. The first is the total number of rows processed at each level of the query plan; this is referred to as the plan's cardinality. The second criterion is the cost model of the search algorithm, which is determined by the operators used in the query. The two criteria are directly related: the better a query's cardinality estimate, the better the cost estimates for the various operators, and ultimately, the faster and more efficient the resulting query plan is deemed to be.

To calculate cardinality, the optimizer considers factors such as indexes, statistics, table constraints, and logical rewrites. But there are factors that can cause SQL Server to consistently calculate inaccurate cardinalities, and they are of particular importance to performance tuning because they often lead to non-optimal query plans. The factors to watch out for include the following:

- Queries with predicates that compare different columns in the same table
- Queries with predicates that use operators referencing any of the following:
  - Columns with missing statistics
  - Out-of-date or non-uniform statistics for a query that seeks a highly selective value set (especially with equal-to (=) operators)
  - Comparisons using the not-equal-to or NOT operators
  - Functions that do not use constants for their arguments
  - Column joins through arithmetic or string concatenation operators
  - Variables whose values can be determined only at run-time

## Excessive or Redundant Recompiles

The third step in troubleshooting CPU-related performance problems is to look for excessive or redundant recompiles. Creating an execution plan uses valuable resources and time, so it's best if SQL Server re-uses existing plans. Compiled query execution plans are stored in the procedure cache, but many factors can cause SQL Server to compile a new plan to satisfy a request rather than re-using an existing plan.

### Ad-hoc Queries

Reuse of ad-hoc query plans is tied to the SQL handle assigned to a statement. In order for an ad-hoc query to have the same SQL handle, the query formatting must be identical; for example, it must have the same number of spaces before and after the query, among other things. SQL Server 2005 is more efficient at caching ad-hoc query plans because it forces parameterization of queries when possible.

One way to guarantee consistent query performance is to force the query optimizer to use a specific plan. SQL Server 2005 introduced a new query hint, USE PLAN, which provides you with full control over the plan used to execute a query. (With earlier versions of SQL Server, you could use query hints such as FORCE ORDER, LOOP JOIN, and KEEP PLAN, but these did not guarantee the plan chosen by the optimizer: factors such as the number of rows returned by operators, statistics, and skewed data could still affect the plan chosen.) Of course, there are restrictions as to when USE PLAN can be and should be used. First, since the query optimizer is typically very adept at choosing the correct plan, the USE PLAN option should be reserved for complex code that has been proven to benefit from a specific plan. Second, only SELECT and SELECT INTO statement query plans can be forced; UPDATE, DELETE, or INSERT statement query plans cannot be forced.



## SET Options

The query optimizer is sensitive to certain SET options, and changing them within compiled code (within the stored procedure, function or trigger) will force the optimizer to recreate the query plan. Changing these options outside the object definition is preferable because it can result in fewer recompiles. The SET options to be careful with are the following:

- ANSI\_NULLS
- ANSI\_PADDING
- ANSI\_WARNINGS
- ANSI\_NULL\_DFLT\_ON
- ANSI\_NULL\_DFLT\_OFF
- ARITHABORT
- CONCAT\_NULL\_YIELDS\_NULL
- DATAFIRST
- DATEFORMAT
- FORCEPLAN
- LANGUAGE
- NO\_BROWSETABLE
- NUMERIC\_ROUNDABORT
- QUOTED\_IDENTIFIER

## Literals Longer Than 8 KB

When a query includes a literal longer than 8 KB, the execution plan will be recompiled for every execution.

## Correctness and Mixing DDL and DML Statements

If the underlying schema referenced by a query changes between statement executions or within a batch execution, the query optimizer will recompile the query plan. Actions like adding a column, dropping a constraint, or creating an index can force the optimizer to reevaluate the most efficient query plan for a statement. To avoid adversely affecting your workload during production hours, schedule DDL changes for a maintenance window, and if possible, do not include them in application code.

## Poor Parallel Query Performance

Poor parallel query performance can also lead to CPU bottlenecks. While evaluating the fastest response time for a query, the query optimizer will elect to execute a statement in parallel if the query's cost exceeds the value specified in the cost threshold for the parallelism option and parallelism has not been disabled. The degree of parallelism used (the number of threads / workers assigned to a query) is decided at execution time. SQL Server will evaluate how many schedulers are underutilized and choose a degree of parallelism that fully utilizes the schedulers available. The maximum degree of parallelism can be limited in two ways: instance-wide, using the max degree of parallelism option (visible using `sp_configure`), or per-query, using the `OPTION (MAXDOP)` hint.

SQL Server's behavior in choosing a degree of parallelism (SQL Server processing a single request using multiple concurrent workers) is in keeping with an often misunderstood design element of SQL Server: all available CPU will be utilized to execute code as quickly as possible. But if a parallel plan is chosen while the system is idle or not using much CPU, a sudden increase in CPU consumption during the execution can lead to a bottleneck.

## Hardware and SQL Server Configuration Issues

If you have checked the factors described above and still believe your system is CPU bound, the final step is to look at configuration issues. Begin by trending the results of the following query:

```
select AVG(runnable_tasks_count) from sys.dm_os_schedulers where status = 'VISIBLE ONLINE'
```

If this query consistently returns a value higher than 1 (and you've already excluded the workload factors described above), you may need to address the available CPU resources.

If your system is not CPU bound, and you don't notice any other clear resource bottlenecks (memory or I/O for example) you should determine whether there is a long work queue for each scheduler. This can be determined by running the following query:

```
select AVG(work_queue_count) from sys.dm_os_schedulers where status = 'VISIBLE ONLINE'
```

If this query consistently returns a value greater than 1, and all other troubleshooting avenues have been pursued, you may benefit from adjusting the `sp_configure max server threads` configuration to make more threads available to SQL Server.

## Troubleshooting CPU Contention Using Quest Tools

### Foglight Performance Analysis for SQL Server

Performance Analysis provides several ways to identify whether a CPU contention issue lies with your SQL Server workload or with the non-SQL Server processes running on the host O/S. And if the root cause does not appear to be other O/S activity, Performance Analysis provides a number of ways to troubleshoot your SQL Server application workload to maximize the use of CPU resources.

#### The Homepage Dashboard

The homepage dashboard has one indicator for your SQL Server workload CPU consumption (figure 17) and another for the CPU consumption of the host O/S (under System Utilization). The value for SQL Server's workload CPU consumption is indicative of the percentage of the overall statement activity that used CPU. This value is always a percentage of the 100% workload activity, split among the various resource categories displayed.

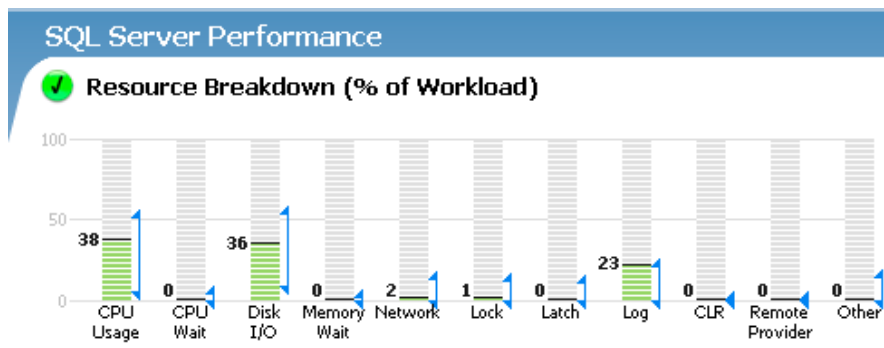
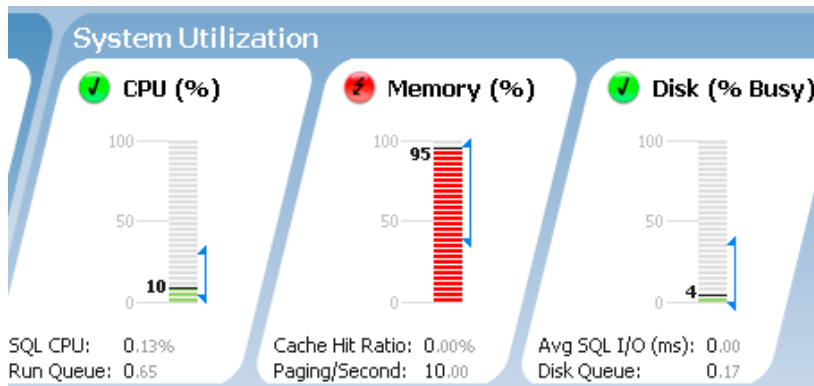


Figure 17—Foglight Performance Analysis for SQL Server CPU utilization counter

The value for Windows CPU utilization is the percentage of the available CPU resources on the machine currently being utilized (figure 18).



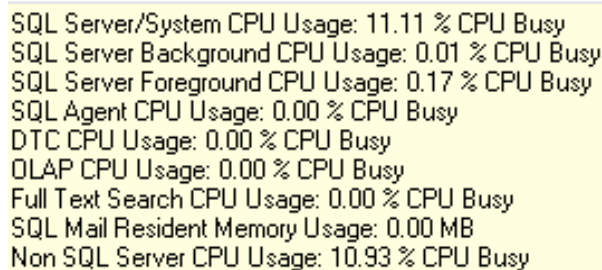


**Figure 18—Foglight Performance Analysis for SQL Server system utilization counter**


When you interpret the information displayed, it is important to understand that the Windows and SQL Workload values can differ; for example, Windows can be using 30% of its available CPU resources, while the SQL workload is predominantly using a different resource. The values differ because the SQL workload information is actually an analysis of the wait events generated by the SQL workload, whereas the Windows CPU information is a system counter displaying the physical CPU activity. The baseline indicators are your guide to understanding whether the amount of CPU being consumed is normal for the period of time displayed. If the amount of CPU consumed deviates from the baseline, the appropriate CPU gauge will change to yellow or red. If the deviation is significant, or if Performance Analysis can offer guidance on tuning your application workload’s use of CPU resources, a flag icon will appear next to the gauge. The flag icon indicates that either a performance advisory or a baseline deviation advisory is available to highlight significant metrics and/or workload trends that should be addressed to maximize the efficiency of CPU-intensive processing.

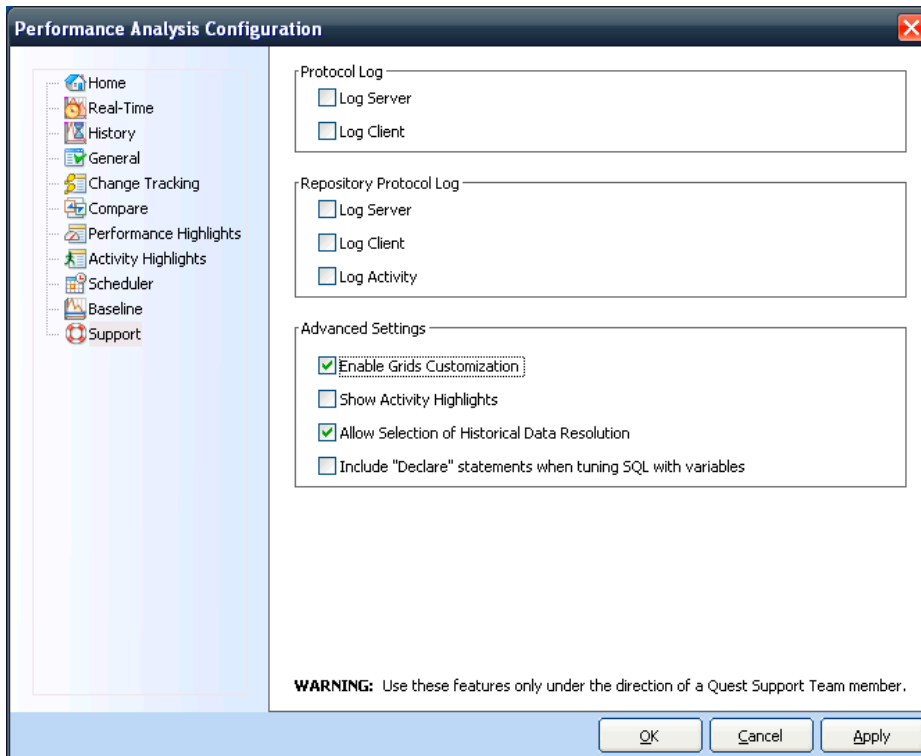
### The Real-time View

The Real-time View displays recent data in two ways: from an instance perspective for up to the last hour, and from a current session perspective for truly real-time data. The Instance View allows you to see your instance workload relative to performance metrics for the time period specified. This view allows you to highlight metrics like SQL Server/System CPU Usage (figure 19) to understand whether the majority of the CPU was used by SQL Server or by other O/S processes.



**Figure 19—Foglight Performance Analysis for SQL Server usage summary**

The value for Non-SQL Server CPU Usage should be evaluated to determine if the majority of the CPU utilized was a result of the SQL server workload or another application. You can add additional metrics to the tabular metric listing by clicking on the Display Options icon , selecting the Support option in the Performance Analysis Configuration dialog, and checking the “Enable Grids Customization” checkbox (figure 20).



**Figure 20—Foglight Performance Analysis for SQL Server configuration options**

To customize the metric listing for a particular screen, including the order of metrics presented, right-click any metric listing table and choose “Select Metrics.” Your selection is sensitive to screen and resource; for example, if you limit the resources displayed to CPU only, only the CPU screen will be affected.

In addition to metric data, the Top Sessions tab will display the most significant resource-consuming sessions during the time period selected (the number of sessions displayed is configurable). If you suspect that a particular Session ID was consuming an inordinate amount of CPU, you may want to sort the list by CPU usage. If you see or suspect that a CPU-intensive session is currently executing, the Current Sessions view will provide you with the information typically available from running `sp_who` or `sp_who2`, or from querying `sysprocesses` or `sys.dm_exec_requests`. Performance Analysis extends basic OEM functionality by providing a summary of all statement activity generated by a particular Session ID, in addition to providing clear visibility into the dynamics of how the application workload used SQL Server resources.

### Performance Advisories

The following Performance Analysis for SQL Server performance advisories can help troubleshoot CPU contention:

- **Compiles/Recompiles**—Determine whether significant CPU contention exists, and, if so, show the transact SQL statements that were observed to be experiencing a high number of recompiles and consumed a large amount of CPU resources. Applications whose SQL code is frequently recompiled will usually benefit from the following:
  - Evaluating the purpose of the statements involved and separating data definition commands from data modification code
  - Addressing out-of-date index statistics
  - Replacing temporary tables with variables or other logic

- **Database File Configuration**—Determine whether significant disk I/O contention exists, and, if so, show how poorly configured database files can lead to increased latch contention within the database, which in turn leads to resource bottlenecks and increased contention within applications. This advisory alerts DBAs to poor database file configuration scenarios for databases suffering from latch contention, including the following:
  - Data and log files configured on the same drive
  - Fewer database files than available CPUs, (with special emphasis on TempDB)
  - Fewer database files than available disk I/O devices
- **Database File Growth**—Determine whether significant disk I/O contention exists, and, if so, show the databases that have been observed to take extents too frequently. This advisory alerts DBAs to databases that have taken frequent extents over a configurable time frame. When SQL Server grows a database file, that file is more prone to corruption and fragmentation, and the process is extremely CPU and I/O intensive.
- **Inefficient Query Plans**—Determine whether significant CPU contention exists, and, if so, illustrate how inefficient query plans can consume excessive CPU resources. Inefficient queries can be forced to be executed in a production environment because of existing database schemas, application requirements, user-enabled report wizards, or other conditions. This advisory alerts DBAs to queries using HASH joins and SORT operations that are resulting in high CPU and I/O consumption.
- **Inefficient or Missing Indexes**—Determine whether significant disk I/O contention exists, and, if so, illustrate how missing or inefficient indexes create disk I/O bottlenecks. DBAs are required to evaluate application SQL code and ensure that the statements executed are as efficient as possible, which generally entails creating indexes to most efficiently extract the data. But if the SQL code changes to either access the tables differently or to select more or different columns from the target tables, the existing indexes may no longer apply. This advisory will illustrate where SQL code is not effectively using existing indexes and where statements are using table scans to gather data.

Whenever one of these performance advisories is generated by Performance Analysis, reviewing the context-sensitive information displayed will pinpoint how you can tune your SQL application code to best utilize CPU resources and optimize performance. Once you have reviewed the information and implemented strategies to manage CPU utilization, Performance Analysis will evaluate whether your strategy has been successful.

### The History View

The History View affords the same view of your SQL Server and O/S environment as the Real-time View, but it includes a baseline context to the SQL workload and to the metric data provided, a consolidation of the advisories generated during a time period, and an OLAP-style drill-down of your SQL workload data for a detailed workload analysis. The History View does not include session (SPID) data. As with the Real-time View, changing the resource to CPU will display only CPU-specific data, so performing a drill-down analysis into your SQL workload will filter any non-CPU-related activity.

## Spotlight on SQL Server Enterprise

Spotlight on SQL Server Enterprise provides several ways to identify whether a CPU contention issue lies with your SQL Server workload or with the non-SQL Server processes running on the host O/S.

### Spotlight Today

When you first open the Spotlight console, Spotlight Today will identify your top server alarms, so you will know immediately whether a CPU-specific condition was observed.

### The Topology (Enterprise) View

Each node (connection) displayed by the Topology View is shown in both a clustered format (see figure 16) and individual node format. The node format (figure 21) provides a visual indication of the health of both SQL Server (the upper arch in figure 21, which in this case is red, indicating a severe problem) and of the underlying Windows O/S (the lower arch, which in this case is blue, indicating an informational message). Hovering over a node will display a pop-up message listing the most recent alarms generated by that connection.



Figure 21—Spotlight on SQL Server Enterprise node view

### The SQL Server Homepage Dashboard

The SQL Server Homepage Dashboard displays information about O/S CPU utilization, in addition to information on the current health of your SQL Server instance. Each column displays information about an architectural component of SQL Server, so you can quickly determine which, if any, aspect of SQL Server is experiencing a bottleneck. Clicking on the CPU gauge will provide a related drill-down to continue your investigation.

### The SQL Server Drill-down Dashboards

When investigating CPU issues in a SQL Server context, you can navigate to the Sessions drill-down (figure 22) and review the currently active sessions to determine whether any SQL Server process is consuming high amounts of CPU.

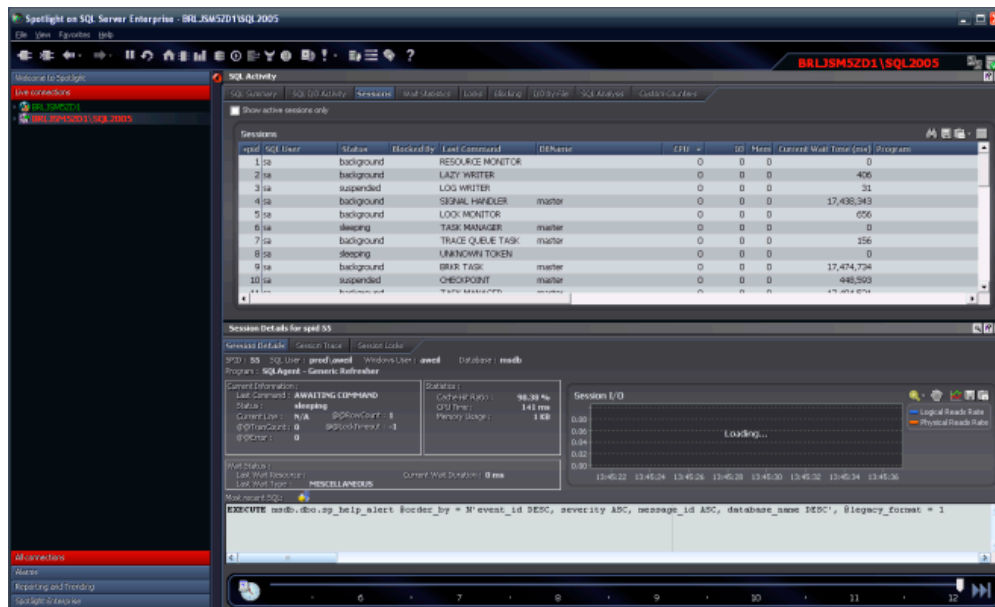


Figure 22—Spotlight on SQL Server Enterprise Sessions drilldown

In addition, you can review the information on the SQL Analysis drill-down to determine whether a particular SQL statement or batch is consuming high amounts of CPU. If SQL Analysis has not been configured, you can override the default (SQL Analysis is disabled by default) and create a trace condition to capture SQL activity within a set of thresholds. SQL Analysis provides the flexibility to define either a continuous rowset trace, or to define a polling server-side trace (which is less resource intensive).

You can also evaluate the data in the Memory drill-down dashboard to evaluate the use of SQL Server's procedure cache. If statements are frequently recompiled, you should see an indication in the Object Types graph. By looking at both the Hit Rate and Use Rate for procedure cache object types (clicking the down arrow in the graph title will let you change the data displayed in the graph), you can determine what types of objects are not being reused, and then leverage the data captured by SQL Analysis to determine how you can follow the steps outlined above for maximizing plan reuse.

In addition to the default metric collection set, Spotlight allows you to define custom counters. It is important to determine whether your SQL Server is CPU-bound (figure 23).



**Figure 23—Spotlight on SQL Server Enterprise SQL Activity drilldown**

Spotlight allows you to define a custom counter to trend the results of the following query:

```
select AVG(runnable_tasks_count)
from sys.dm_os_schedulers
where status = 'VISIBLE ONLINE'
```

If you create a custom counter called “CPU Runnable Tasks” using the query above, and it consistently returns a value higher than 1, and if you’ve already excluded workload factors, you may need to address the CPU resources available on the machine.

### The Windows Homepage Dashboard

The Windows Homepage Dashboard assesses the health of your Windows O/S with a column dedicated to the major aspects of the Windows architecture. High-level information includes the number of CPU cores on the machine, their clock speed, and their type. The CPU (figure 24) column enables you to immediately determine whether high CPU utilization or a large processor queue is creating problems for Windows.

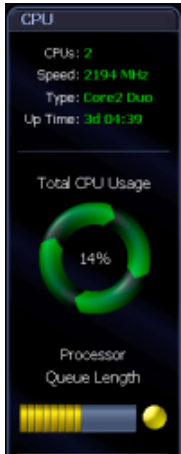


Figure 24—Spotlight on SQL Server Enterprise CPU column

In many cases, the CPU utilization will tell you whether Windows is experiencing high CPU utilization, but if there is no Processor Queue, the O/S is not backlogged with requests.

### The Windows Drill-down Dashboards

Clicking on the CPU gauge will take you to the Processes drill-down so you can review a list of the most CPU-intensive processes (figure 25). Click on a particular process to see detailed information about it, including Processor Utilization History, which can help you determine recent CPU activity.



Figure 25—Spotlight's Windows CPU drilldown

If a large percentage of the CPU being consumed on your server is from non-SQL Server activity, the process drill-down will identify your top CPU consumers.

Using Spotlight on SQL Server Enterprise, real-time data and data up to seven days in the past is easily visible using the Playback feature (via the slider at the bottom of the screen).

## I/O Bottlenecks

Your SQL Server can perform only as well as the underlying I/O subsystem. The constant process of moving database pages in and out of the buffer pool and flushing log records to disk generates significant I/O traffic. (The other significant cause of I/O activity, TempDB, is addressed in its own section later in this document.) There are two types of I/O that a DBA needs to tune:

- Sequential, or (typically) log file I/O
- Random, or data file I/O

Because these are two distinct types of I/O activity, the generally accepted best practice is to split data and log files onto different physical disks. When either process is slow, however, users will complain of all manner of application errors or simply a general slowness in response time.

## Using Performance Counters

Typically, DBAs are trained to focus on a specific subset of performance counters to investigate I/O bottlenecks. In particular, they tend to look at the following counters:

- **Physical Disk Object: Avg. Disk Queue Length:** This metric reveals the average number of physical I/O (read and write) requests waiting for a specific disk. While this can be a useful metric, it cannot be looked to exclusively to indicate an I/O bottleneck; similarly, there is no generally accepted “good” value that indicates an I/O subsystem problem. The trend for SQL Server to use RAID and SAN storage in lieu of standard-issue physically attached single disks further complicates matters: there can be a significant disk queue on parity devices in RAID configurations, and SAN disk queue lengths can reach in the 100s while the application shows no signs of slowness.
- **Avg. Disk sec/Transfer:** This metric reveals the complete round trip time of a request, so it is a direct measure of disk response time, including queue time. This is arguably a more telling metric when using SAN storage or other storage that is not of the traditional physically attached variety, and it can be used together with Avg. Disk sec/Read and Avg. disk sec/Write to identify a bottleneck.
- **Avg. Disk Sec/Read and Avg. Disk Sec/Write:** These metrics reveal the average number of seconds required for a read or write of data from or to the disk. Unlike Disk Queue Length, these metrics have generally accepted “good” values:
  - Less than 10 ms—Excellent
  - Between 10—20 ms—OK
  - Between 20—50 ms—Investigate
  - Greater than 50 ms—Serious I/O bottleneck
- **Physical Disk: %Disk Time:** This metric reveals how busy the selected disk is servicing the current workload. Typically, values greater than 50 percent require investigation to rule out a performance bottleneck.
- **Avg. Disk Reads/Sec and Avg. Disk Writes/Sec:** These metrics reveal the rate of physical read and write operations. Values up to 85 percent are generally acceptable for this metric; however, once that threshold is exceeded, access time will increase exponentially.



Whenever you collect any of these performance metrics, it is crucial to bear in mind that the numbers are averages and therefore need to be trended to identify a performance bottleneck. In addition, if RAID devices are implemented, the following formulas should be used to alter the data returned by the “raw” counters:

- **Raid 0:** I/Os per disk = (reads + writes) / number of disks
- **Raid 1:** I/Os per disk = [reads + (2 \* writes)] / 2
- **Raid 5:** I/Os per disk = [reads + (4 \* writes)] / number of disks
- **Raid 10:** I/Os per disk = [reads + (2 \* writes)] / number of disks

When you have identified an I/O bottleneck, you can address it by doing one or more of the following:

- **Check for memory pressure and investigate SQL Server’s memory configuration.** If SQL Server has been configured with insufficient memory, it will incur more I/O overhead. For more information, see the *Memory Pressure section later* in this document.
- **Investigate your server hardware:** You may need to effectively increase the I/O bandwidth available to SQL Server. This can be accomplished in two ways (usually together with a system administrator and/or storage administrator):
  - Add more physical drives and/or upgrade to faster drives.
  - Add faster or additional I/O controllers.

Of course, SQL Server DBAs usually must exhaust all available avenues before upgrading hardware on a machine. While this is not the fastest alternative, in many cases tuning your application workload can greatly reduce the amount of I/O contention on a system. Wait Event Analysis and query tuning can go a long way in successfully reducing the I/O load on your system.

## Wait Event Analysis

Wait Event analysis allows you to use traditional system views and DMVs to understand how the current SQL Server workload is performing. By investigating LATCH waits, specifically PAGEIOLATCH\_x waits, you can understand the physical I/O waits (not buffer reads/writes) imposed by your applications. Whenever a request cannot be satisfied by the buffer pool, SQL Server will issue an asynchronous I/O request for the data. If that request must wait to be satisfied, a PAGEIOLATCH\_EX or PAGEIOLATCH\_SH event will be generated.

Performance tuning using waits and queues is not a simple process. You must not only understand individual wait events and what they indicate, but you must also understand the concept of a signal wait. You may see a specific amount of time associated with a particular wait event, but until you verify the signal wait time is not increasing, you cannot be sure that SQL Server has successfully scheduled a worker for the request.

Typically, a review of wait events will lead to an investigation of statement query plans. When you look at a SQL Server execution plan, the I/O related information displayed does not account for physical writes. This is because of the nature of SQL Server processing; database physical writes occur when transactions are committed and a checkpoint is triggered or the SQL Server lazy writer writes information to disk. When you tune queries to address I/O-related wait events and to reduce logical I/O activity, your SQL Server workload will more effectively leverage the buffer pool and put less strain on the physical disk I/O subsystem.



## Troubleshooting I/O Bottlenecks Using Quest Tools

### Foglight Performance Analysis for SQL Server

When you investigate I/O bottlenecks with Performance Analysis, there are several ways to identify whether the issue lies with your SQL Server workload or with the non-SQL Server processes running on the host O/S.

#### The Homepage Dashboard

The Homepage Dashboard has one indicator for your SQL Server workload I/O consumption (under SQL Server Performance, figure 26)

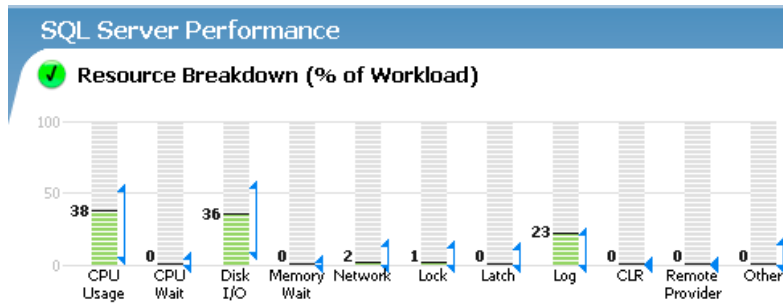


Figure 26—Foglight Performance Analysis for SQL Server I/O indicator

and another for the I/O consumption of the host O/S (under System Utilization, figure 27).

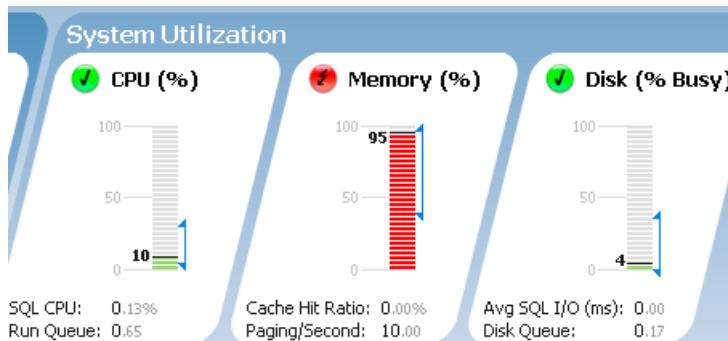


Figure 27—Foglight Performance Analysis for SQL Server system utilization

The value for SQL Server's workload I/O consumption indicates the percentage of the overall statement activity that used I/O. This value is always a percentage of the 100% workload activity, split among the various resource categories displayed. The value for the Windows I/O utilization is actually a reflection of the %Busy of the busiest disk configured for the O/S during the time range specified.

It is important to understand that the Windows and SQL Workload values can differ; for example, the busiest configured disk can be 80% busy, while the SQL workload is predominantly using a different resource. The values differ because the SQL workload information is actually an analysis of the wait events generated by the SQL workload, whereas the Windows I/O information is a system counter displaying their physical I/O activity. The baseline indicators are your guide to understanding whether the amount of I/O being consumed is normal for the period of time displayed. If the amount of I/O consumed deviates from the baseline, the appropriate I/O gauge will turn yellow or red to indicate the deviation. If the deviation is significant, or if Performance Analysis can offer guidance on tuning your application workload's use of I/O resources, a flag icon will appear next to the gauge. The flag icon indicates that either a performance advisory or a baseline deviation advisory is available to highlight significant metrics and/or workload trends that should be addressed to maximize the efficiency of I/O-intensive processing.

## The Real-time View

The Real-time View displays recent data in two ways: from an instance perspective for up to the last hour, and from a current session perspective for truly real-time data. The Instance View allows you to see your instance workload relative to performance metrics for the time period specified. This view allows you to highlight metrics like Total Physical I/O Operations (Figure 28) to understand whether the majority of the I/O was generated by SQL Server or by other O/S processes.


```
Total Physical I/O Operations: 81,016,227.00
Total Physical Reads: 27,309,187.00
Total Physical Writes: 53,707,040.00
```

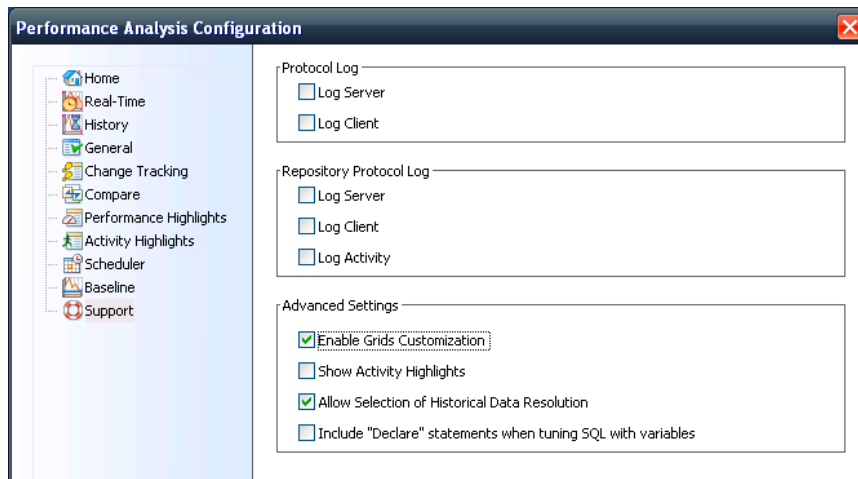
**Figure 28—Total Physical I/O Operations**

The value for Total Physical I/O Operations should be compared to Total SQL Server Physical I/O Operations (Figure 29) to determine whether the majority of the I/O generated was a result of the SQL server workload or another application.

```
SQL Server Physical I/O Operations: 34,070,058.00 Page Operations
Pages Read: 12,345,141.00 Pages
Pages Written: 21,724,917.00 Pages
```

**Figure 29—SQL Server Physical I/O**

You can add additional metrics to the tabular metric listing by clicking on the Display Options icon , selecting the Support option in the Performance Analysis Configuration dialog, and checking the “Enable Grids Customization” checkbox.



**Figure 30—Performance Analysis Configuration options**

To customize the metric listing for a particular screen, including the order of metrics presented, right-click any metric listing table and choose “Select Metrics.” The selection you make is sensitive to screen and resource; for example, if you limit the resources displayed to I/O only, only the I/O screen will be affected.

In addition to metric data, the Top Sessions tab displays the most significant resource-consuming sessions during the time period selected (the number of sessions displayed is configurable). If you suspect that a particular Session ID was consuming an inordinate amount of I/O, you may want to sort the list by CPU usage. If you see or suspect that a I/O - intensive session is currently executing, the Current Sessions view will provide you with the information typically available from running `sp_who`, `sp_who2`, or from querying `sysprocesses`, or `sys.dm_exec_requests`. Performance Analysis extends basic OEM functionality by providing a summary of all statement activity generated by a particular Session ID, in addition to giving clear visibility into the dynamics of how the application workload used SQL Server resources.

## The History View

The History View affords the same view of your SQL Server and O/S environment as the Real-time View, but it includes a baseline context to the SQL workload and to the metric data provided, a consolidation of the advisories generated during a time period, and an OLAP-style drill-down of your SQL workload data for a detailed workload analysis. The History View does not include session (SPID) data. As with the Real-time View, changing the resource to I/O will display only I/O-specific data, so performing a drill-down analysis into your SQL workload will filter any non-I/O-related activity.

## Performance Advisories

The following Performance Analysis for SQL Server performance advisories can help troubleshoot I/O bottlenecks:

- **Database File Configuration**—Determine whether significant disk I/O contention exists, and, if so, show how poorly configured database files can lead to increased latch contention within the database, which in turn can lead to resource bottlenecks and increased contention within applications. This advisory alerts DBAs to a number of poor database file configuration scenarios for databases suffering from latch contention, including the following:
  - Data and log files configured on the same drive
  - Fewer database files than available CPUs, with special emphasis on TempDB
  - Fewer database files than available disk I/O devices
- **Database File Growth**—Determine whether significant disk I/O contention exists, and, if so, show the databases that have been observed to take extents too frequently. This advisory alerts DBAs to databases that have taken frequent extents over a configurable time frame. When SQL Server grows a database file, that file is more prone to corruption and fragmentation, and the process is extremely CPU and I/O intensive.
- **Inefficient Query Plans**—Determine whether significant CPU contention exists, and, if so, illustrate how inefficient query plans can consume excessive CPU resources. Existing database schemas, application requirements, user-enabled report wizards, and other conditions can force inefficient queries to be executed in a production environment; this advisory alerts DBAs to queries using HASH joins and SORT operations that are resulting high CPU and I/O consumption.
- **Inefficient or Missing Indexes**—Determine whether significant disk I/O contention exists, and, if so, illustrate how missing or inefficient indexes create disk I/O bottlenecks. DBAs are required to evaluate application SQL code and ensure that the statements executed are as efficient as possible, which generally entails creating indexes to most efficiently extract the data. But if the application SQL code changes to either access the tables differently or to select more or different columns from the target tables, the existing indexes may no longer apply. This advisory illustrates where SQL code is not effectively using existing indexes and where statements are using table scans to gather data.
- **Internal Cache Latch Contention**—Determine whether significant internal cache latch contention exists, and if so, identify where the majority of the contention exists. Internal cache latches can be used for a variety of things; possibly the most common case is contention on internal caches (not buffer pool pages), especially when using heaps, text, or both. If solving LOG and PAGELATCH\_UP contention does not help, internal cache latch contention can usually best be treated by partitioning data.
- **Log Waits**—Determine whether significant log waits were observed, and if so, show how a number of factors can contribute to SQL Server logging slowdowns.

Whenever one of these performance advisories is generated by Performance Analysis, reviewing the context-sensitive information displayed will pinpoint how you can tune your I/O accesses to be more efficient. Once you have reviewed the information and implemented strategies to adjust your applications' access to the underlying I/O devices, Performance Analysis will assess whether your strategy has been successful.

## Spotlight on SQL Server Enterprise

Spotlight on SQL Server Enterprise provides several ways to identify whether the causes of I/O bottlenecks lie with your SQL Server workload or with the non-SQL Server processes running on the host O/S.

### Spotlight Today

When you first open the Spotlight console, Spotlight Today will identify your top server alarms, so you will know immediately whether an I/O-specific condition was observed.

### The Topology (Enterprise) View

Each node (connection) displayed by the topology view provides a visual indication of the health of both SQL Server (the upper arch, which in figure 31 is red, indicating a severe problem) and of the underlying Windows O/S (the lower arch in figure 31, which is blue, is indicating an informational message).



Figure 31—Spotlight on SQL Server Enterprise node view

Hovering over a node will display a pop-up message listing the most recent alarms generated by that connection.

### The SQL Server Homepage Dashboard

The SQL Server Homepage Dashboard displays information about your O/S I/O utilization and information about the current health of your SQL Server instance (figure 32). Each column displays information about an architectural component of SQL Server, so you can quickly determine which, if any, aspect of SQL Server is experiencing a bottleneck. Clicking on the Data Files gauge or the Log Files gauge will provide a related drill-down to continue your investigation. There are indications in the I/O column for:

- The number of databases defined for this instance (severity relative to days since last backup)
- Data Files
  - The number of file groups defined
  - The number of individual data files defined
  - The aggregated size of all database data files for this instance
  - The total percentage in use for all database data files for the instance
- Log Files
  - The number of log files defined
  - The aggregated size of all database log files for this instance
  - The total percentage in use for all database log files for this instance
- Disk Queue Length

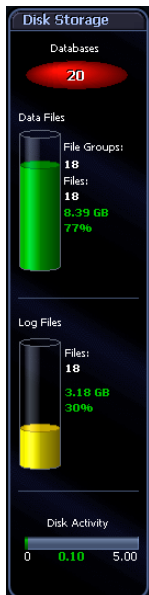


Figure 32—Spotlight on SQL Server Enterprise disk activity column

### The SQL Server Drill-down Dashboards

When investigating I/O bottlenecks in a SQL Server context, you can navigate to the Sessions drill-down and review the currently active sessions to determine whether any SQL Server process is consuming high amounts of I/O. In addition, you can review the information on the SQL Analysis drill-down to determine whether a particular SQL statement or batch is consuming high amounts of I/O (figure 33).



Figure 33—Spotlight on SQL Server Enterprise SQL Activity drilldown

If SQL Analysis has not been configured, you can override the default (SQL Analysis is disabled by default) and create a trace condition to capture SQL activity within a set of thresholds. SQL Analysis provides the flexibility to define either a continuous rowset trace, or to define a polling server-side trace (which is less resource intensive).

You can also evaluate which files are experiencing the highest I/O load. Once you find the sessions or SQL statement activity generating the most significant I/O, the I/O by File display will help you determine whether moving any of your SQL Server database data or log files to another I/O device might be warranted, or if partitioning the data might help you manage the I/O activity generated by your applications.

In addition to the default metric collection set, Spotlight allows you to define custom counters. When you are investigating an I/O-related condition, it is important to determine whether your SQL Server or your disk storage subsystem is I/O-bound, by investigating latencies. Spotlight allows you to define custom counters to trend the results of the following queries:

Disk I/O Pending (SQL Server Custom Counter):

```
select [pending_disk_io_count]
from sys.dm_os_schedulers
```

Disk Avg. sec/Transfer (Windows Custom Counter):

```
Select AvgDiskSecPerTransfer from Win32_PerfFormattedData_PerfDisk_PhysicalDisk
```

Disk Avg. sec/Read (Windows Custom Counter):

```
Select AvgDiskSecPerRead from Win32_PerfFormattedData_PerfDisk_PhysicalDisk
```

Disk Avg. sec/Write (Windows Custom Counter):

```
Select AvgDiskSecPerWrite from Win32_PerfFormattedData_PerfDisk_PhysicalDisk
```

If you create these custom counters and SQL Server shows that it is consistently waiting to process I/O requests, or that the transfer times for the disk storage subsystem are unusually long, you may need to address the physical disks configured on your machine.

### **The Windows Homepage Dashboard**

The Windows Homepage Dashboard assesses the health of your Windows O/S with a column dedicated to the major aspects of the Windows architecture (figure 34). It provides high-level information, including the number of I/O devices on the machine (including page file information), their capacity, and the percentage of utilization. The I/O column will allow you to immediately determine if high I/O utilization is creating problems for Windows. If your system is experiencing an I/O bottleneck brought about by high paging (when Windows uses the page file to satisfy memory requests) that activity will generate increased CPU utilization. If this is the case (CPU activity is high but you're noticing I/O bottlenecks in your SQL Server workload), you should investigate Windows' page file usage (click on the Page Faults or Paging icon on the home page) to determine whether the page files defined on your I/O devices where SQL Server files reside are leading to a bottleneck.

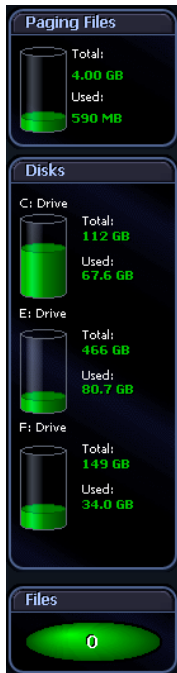


Figure 34—Spotlight's Windows I/O column

### The Windows Drill-down Dashboards

Clicking through to to Disk Information drill-down will allow you to review the logical and physical I/O activity on your SQL Server (figure 35).



Figure 35—Spotlight's Windows disk activity drilldown



Here you can assess the root cause of the I/O bottleneck, including the I/O by disk, the transfer time by disk, and indications of the load imposed on each disk. Related information about the configuration of each disk is provided so you can understand the specifics of the I/O devices displaying the heaviest load or the most significant latencies. Real-time data and data up to seven days in the past is easily visible using the Playback feature (via the slider at the bottom of the screen).

## Memory Pressure

### Types of Memory

Many SQL Server DBAs make the mistake of addressing SQL Server memory as a single entity. In fact, there are two distinct types of memory available to SQL Server and Windows processes:

- **Virtual Address Space (VAS):** Each process on a Windows machine has its own [Virtual Address Space \(VAS\)](#). VAS size depends on the architecture of the server (32- or 64-bit) and the operating system installed.
- **Address Windowing Extensions (AWE):** The [Address Windowing Extensions \(AWE\)](#) API allows 32-bit applications to manipulate physical memory beyond the 32-bit address limit. While AWE is technically not necessary 64-bit servers it is present, but the correct terminology on the 64-bit platform is “locked pages.” This memory is unique because it cannot be paged out, a feature that can benefit applications when it is implemented correctly. When this feature is implemented incorrectly, allocating AWE memory to an application (locking pages in memory) can starve other applications on the system.

Note: To leverage the AWE mechanism, the account running SQL Server must be granted the Lock Pages in Memory privilege.

When troubleshooting an environment that leverages AWE, you must be aware of two important points:

1. AWE memory allocations are not reported by Task Manager nor by the performance counter Process Private Bytes.
2. Only database pages can leverage AWE through the buffer pool.

### Relationship between O/S Architecture and SQL Server Architecture

Before you launch an investigation related to memory pressure, it is important to understand the relationship between the O/S architecture and the SQL Server architecture running on it. The following table outlines the maximum memory support options for different configurations of SQL Server 2005.

Configuration	VAS	Max physical memory	AWE/locked pages
32-bit SQL Server and Windows	2 GB	64 GB	Yes
IBID with /3GB boot parameter	3 GB	16 GB	Yes
32-bit SQL Server on Windows x64 (WOW)	4 GB	64 GB	Yes
32-bit SQL Server on Windows IA64 (WOW)	2 GB	2 GB	No
64-bit SQL Server on Windows x64	8 TB	1 TB	Yes
64-bit SQL Server on Windows IA64	7 TB	1 TB	Yes



## Types of Memory Pressure

Memory pressure is a condition created when a limited amount of memory is available. There are different types of memory pressure, including:

- **Internal:** contention among elements of SQL Server's architecture
- **External:** contention among other Windows processes
- **Physical:** contention for physical memory (RAM)
- **Virtual:** contention for logical memory (page file)

SQL Server responds differently to different types of memory pressure:

- **Internal**
  - a. **Physical Memory Pressure:**
    - **Symptoms:** SQL Server will redistribute memory between internal components, which can cause a decrease in page life expectancy and lead to blocking locks or deadlocks.
    - **Troubleshooting:** Identify the major memory consumers within SQL Server, check SQL Server memory configurations, and address the application workload.
  - b. **Virtual Memory Pressure:**
    - Applications loading DLLs into MemToLeave or a high number of threads may lead to SQL server releasing VAS regions and/or shrinking the buffer pool or internal caches.
    - **Troubleshooting:** Identify the major memory consumers within SQL Server, check SQL Server memory configurations, and address the application workload.
- **External**
  - a. **Physical Memory Pressure:**
    - **Symptoms:** Windows will trim working sets, which will generally slow performance and may cause SQL Server to clear internal caches.
    - **Troubleshooting:** Identify and eliminate the major memory consumers and/or add more memory.
  - b. **Virtual Memory Pressure:**
    - **Symptoms:** Because Windows cannot allocate memory using the pagefile, memory allocation errors and severe slowdowns will ensue.
    - **Troubleshooting:** Identify and eliminate the major memory consumers and/or add increase the page file size.

## Troubleshooting Memory Pressure Using Quest Tools

### Foglight Performance Analysis for SQL Server

#### The Homepage Dashboard

The Homepage Dashboard provides one indicator for your SQL Server workload latch and memory waits (figure 36) and another for the memory consumption of the host O/S (figure 37).

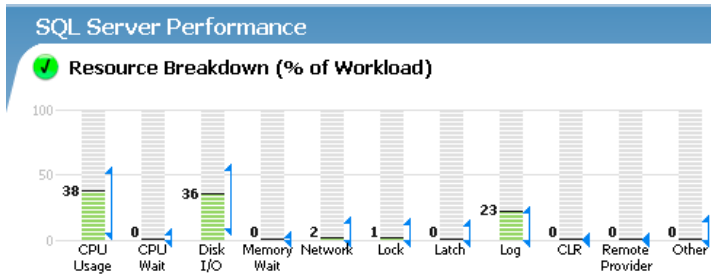


Figure 36—Foglight Performance Analysis for SQL Server's SQL Server performance indicator

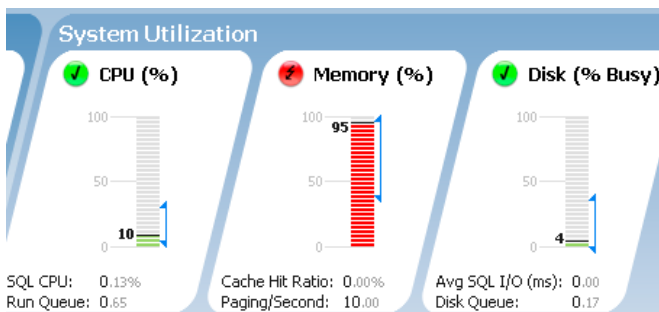


Figure 37—Foglight Performance Analysis for SQL Server's System Utilization counter

The values for SQL Server's workload latch and memory wait indicate the percentage of the overall statement activity that generated latch waits or waits for memory. These values are always a percentage of the 100% workload activity, split among the various resource categories displayed. The value for the Windows Memory utilization is the percentage of the available physical memory on the machine currently being utilized.

It is important to understand that the Windows and SQL Workload values can differ; for example, Windows can be using 80% of its available physical memory, while the SQL workload is predominantly using a different resource. The baseline indicators are your guide to understanding whether the amount of memory being consumed is normal for the period of time displayed. If the amount of memory consumed deviates from the baseline, the appropriate gauge will turn yellow or red to indicate the deviation. If the deviation is significant, or if Performance Analysis can offer guidance on tuning your application workload's use of memory-related resources, a flag icon will appear next to the gauge. The flag icon indicates that either a performance advisory or a baseline deviation advisory is available to highlight significant metrics and/or workload trends that should be addressed to maximize workload processing.

The performance advisories available for tuning latch and memory utilization are:

- **Buffer Latch Contention**—Determine whether significant buffer latch contention exists, and if so, show high buffer latch waits as an indication of I/O bottlenecks and hot pages. Since buffer latching is generally not directly related to I/O contention, this advisory can be sensitive to the amount of memory available to SQL Server.
- **Internal Cache Latch Contention**—Determine whether significant internal cache latch contention exists, and if so, identify where the majority of the contention exists. Internal cache latches can be used for a variety of things; possibly the most common case is contention on internal caches (not buffer pool pages), especially when using heaps, text, or both. If solving LOG and PAGELATCH\_UP contention does not help, internal cache latch contention can usually best be treated by partitioning data.
- **Memory Pressure**—Determine whether significant memory pressure exists, and if so, show how:
  - External memory pressure can impact SQL Server performance. Many DBAs and their managers do not understand the impact on SQL Server performance of improperly configuring virus checking software, installing SQL Server on an Exchange server, and so on.
  - SQL Server did not have enough memory to function optimally. If SQL Server cannot allocate enough memory to its buffer caches, page life expectancy decreases and system-wide paging tends to increase.


### The Real-time View

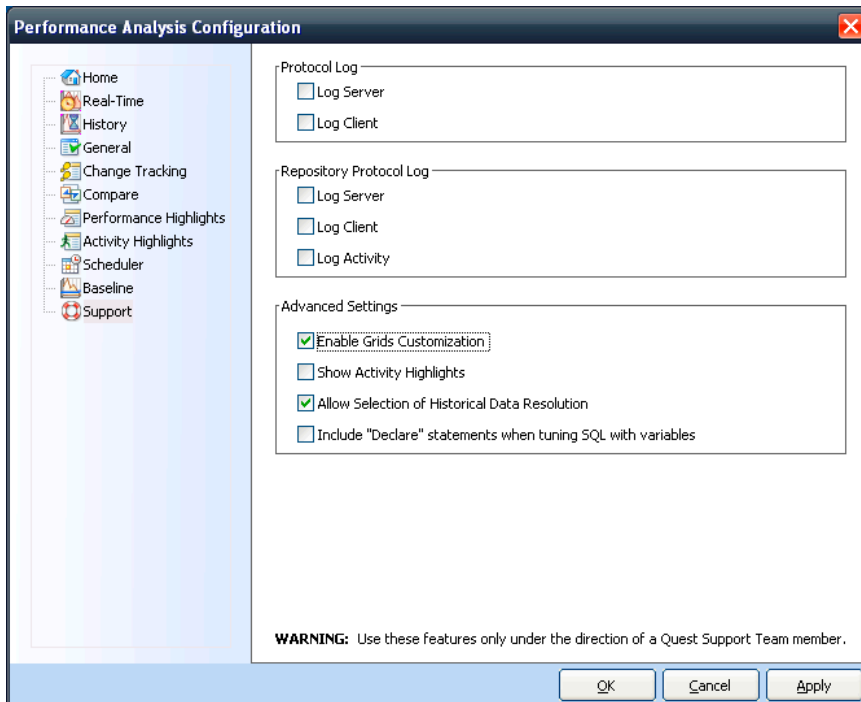
The Real-time View displays recent data in two ways: from an instance perspective for up to the last hour, and from a current session perspective for truly real-time data. The Instance View allows you to see your instance workload relative to performance metrics for the time period specified. This view allows you to highlight metrics like Memory Utilization to understand whether the majority of the available memory was used by SQL Server or by other O/S processes (figure 38).

```
Memory Utilization: 3,364.65 MB
Machine RAM: 3,582.04 MB
SQL Server Resident Memory Usage: 1,348.00 MB
SQL Agent Resident Memory Usage: 0.00 MB
DTC Resident Memory Usage: 4.06 MB
OLAP Resident Memory Usage: 0.00 MB
Full Text Search Resident Memory Usage: 4.28 MB
Non SQL Server Resident Memory Usage: 2,008.31 MB
```

**Figure 38—Foglight Performance Analysis for SQL Server’s memory utilization summary**

The value for Non-SQL Server Memory Usage should be evaluated to determine whether most of the memory utilized was a result of the SQL server workload or another application.

You can add additional metrics to the tabular metric listing by clicking on the Display Options icon , selecting the Support option in the Performance Analysis Configuration dialog, and checking the “Enable Grids Customization” checkbox (figure 39).



**Figure 39—Foglight Performance Analysis for SQL Server’s configuration options**

To customize the metric listing for a particular screen, including the order of metrics presented, right-click any metric listing table and choose “Select Metrics.” The selection you make is sensitive to screen and resource, so, for example, if you limit the resources displayed to Memory only, only the Memory screen will be affected. To better identify memory pressure situations, add Page Life Expectancy and Total Server Memory (the amount of memory SQL Server is currently consuming) versus Target Server Memory (the amount of memory SQL Server requires to perform optimally); this will enable you to check for noticeable drops in the page life expectancy and instances where the target server memory is higher than total server memory.

In addition to metric data, the Top Sessions tab displays the most significant resource-consuming sessions during the time period selected (the number of sessions displayed is configurable). If you suspect that a particular Session ID was consuming an inordinate amount of memory, sort the list by memory usage. If you see or suspect that a memory-intensive session is currently executing, the Current Sessions view will provide you with the information typically available from running `sp_who` or `sp_who2`, or from querying `sysprocesses` or `sys.dm_exec_requests`. Performance Analysis extends basic OEM functionality by providing a summary of all statement activity generated by a particular Session ID, in addition to giving clear visibility into the dynamics of how the application workload used SQL Server resources.

### The History View

The History View affords the same view of your SQL Server and O/S environment as the Real-time View, but it includes a baseline context to the SQL workload and to the metric data provided, a consolidation of the advisories generated during a time period, and an OLAP-style drill-down of your SQL workload data for a detailed workload analysis. The History View does not include session (SPID) data. As with the Real-time View, changing the resource to memory will only display memory-specific data, so performing a drill-down analysis into your SQL workload will filter any non-memory-related activity.

## Spotlight on SQL Server Enterprise

### Internal Memory Pressure

#### The SQL Server Homepage Dashboard

The SQL Server Homepage Dashboard displays information about the memory utilization of both your SQL Server and the host O/S (figure 40).



Figure 40—Spotlight on SQL Server Enterprise homepage dashboard

Each column displays information about an architectural component of SQL Server, so you can quickly determine which, if any, aspect of SQL Server is experiencing a bottleneck. Clicking on the Memory gauge will take you to the Memory Summary drill-down, while clicking on the Page Life Expectancy gauge will take you to the Buffer Cache dashboard where you can review the performance of the SQL Server buffer cache.

#### The SQL Server Drill-down Dashboards

When investigating internal memory pressure, you have to determine how much memory is consumed by the various memory areas within SQL Server (figure 41).



Figure 41—Spotlight on SQL Server Enterprise memory display

Normally, the buffer pool accounts for the most of the memory committed by SQL Server. Clicking on the Buffer Cache tab takes you to the Buffer Cache dashboard, which shows the contents of the buffer pool in addition to a breakdown of page allocations (figure 42).



**Figure 42—Spotlight on SQL Server Enterprise Buffer Cache dashboard**

If you see a high percentage of stolen pages, SQL Server is using pages from the buffer pool to satisfy other memory requests. One way you can determine whether internal memory pressure is being caused by components that use SQL Server memory outside the buffer pool is to create the following two custom counters:

“Bufferpool Commit Target (MB)”, defined as:

```
select bpool_commit_target/128 from sys.dm_os_sys_info
```

and “Bufferpool Committed (MB)”, defined as:

```
select bpool_committed/128 from sys.dm_os_sys_info
```

When the target value is above the committed value, SQL Server will attempt to obtain additional memory. When the reverse is true, the bufferpool will shrink. To take into account AWE-enabled memory, add another custom counter, “AWE Memory (MB)”, defined as:

```
select SUM([awe_allocated_kb])/1024 from [sys].[dm_os_memory_clerks] where [type]='MEMORYCLERK_SQLBUFFERPOOL'
```

To understand how memory pressure has affected your SQL Server over time, you can use the Reporting and Trending Views to run a SQL Server General Statistics report (figure 43).



**Figure 43—Spotlight on SQL Server Enterprise’s memory drilldown**

By selecting an instance and timeframe, information on hit ratios, page life expectancy, and SQL Server Total Server Memory (the amount of memory SQL Server is currently consuming) versus Target Server Memory (the amount of memory SQL Server requires to perform optimally), you can check for noticeable drops in the page life expectancy or instances where the target server memory is higher than total server memory, which can help you identify a memory pressure situation.

## **External Memory Pressure**

### **The Windows Homepage Dashboard**

The Windows Homepage Dashboard assesses the health of your Windows O/S with a column dedicated to the major aspects of the Windows architecture (figure 44). It provides high-level information such as the amount of physical memory installed and the virtual memory configured on the machine, as well as operating system hit ratios. The Memory column will allow you to immediately determine if high memory utilization or a low hit rate are creating problems for Windows. Physical versus Virtual memory utilization will tell you whether Windows is experiencing physical or virtual memory pressure by clearly identifying the used versus free memory in each category. Low available physical memory is low indicates physical memory pressure and usually exists together with high virtual memory usage. To determine the root cause of high utilization, click any of the icons in the memory column to see a related-drill down for a more detailed diagnosis.





Figure 44—Spotlight's Windows O/S information

### The Windows Drill-down Dashboards

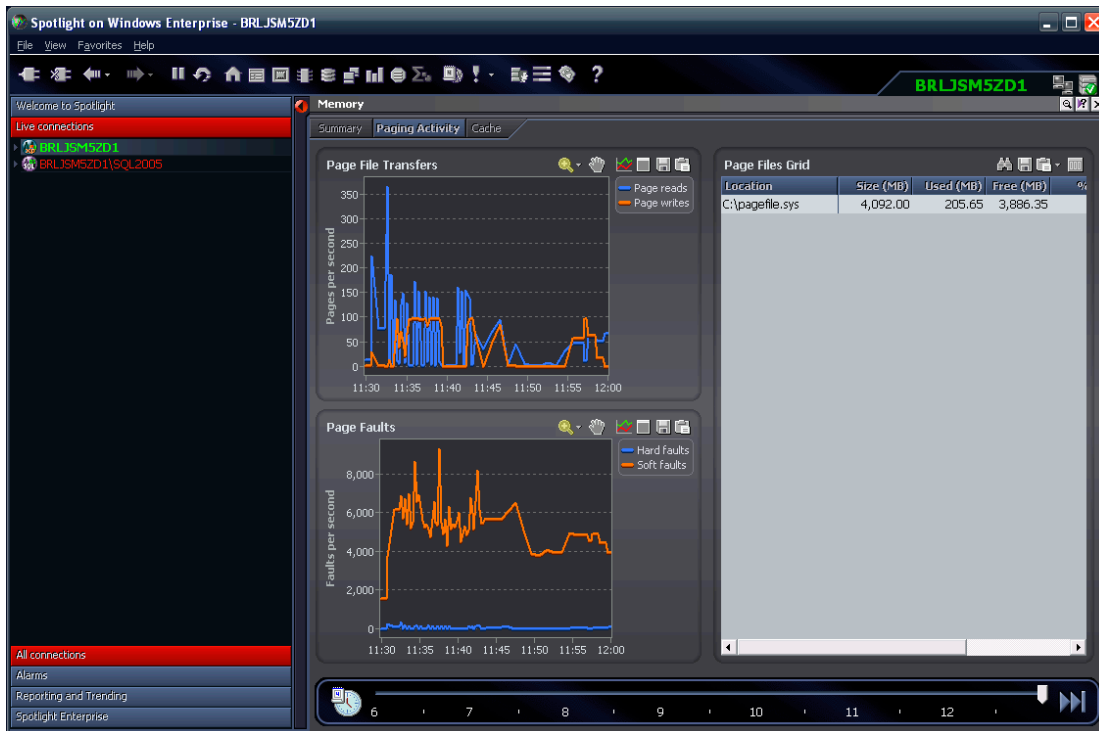
Clicking on the Memory gauges will take you to the Processes drill-down so you can review a list of the most memory-intensive processes (figure 45).




Figure 45—Spotlight's Windows Processes drill down



When you click on a particular process, Spotlight will display detailed information about that process, including memory usage (which correlates to Windows counter Process: Working Set Size) and virtual memory size (which correlates to Windows counter Process: Private Bytes), which can help you determine whether memory pressure is present; if the values for Memory Usage and Virtual Memory Size are nearly equivalent, memory pressure is not present for that process. In order to fully understand memory pressure conditions, your investigation must include the Windows pagefile. You must determine if the page files are configured with a sufficient amount of space to accommodate your applications' memory allocations. To check this, check the Memory Summary dashboard (figure 46) and the Virtual Memory Y-axis (this correlates to the Windows virtual commit charge limit, which is essentially the maximum potential pagefile usage without extending the current pagefile). If the "In Use" value (which correlates to the Windows Total Commit Charge, which is the potential for page file use) is high, page file space may be running low.



**Figure 46—Spotlight’s Windows Memory Summary dashboard**

Equivalent information is visible in the table view under Paging Activity, which shows the Paging File Total, % Used, and Peak % (the most pagefile in use since the last reboot). If you notice that the Peak % value is high, you can check the System Event Log in the Event Logs dashboard (  ) for page file growth events or notifications of Windows “running low on virtual memory.” If these messages exist, you may need to increase the size of your page files. In contrast, high values for % Used indicate an over-commitment of physical memory. Real-time data and data up to seven days in the past is easily visible using the Playback feature (via the slider at the bottom of the screen).

# TempDB Troubleshooting

Troubleshooting performance problems related to TempDB can be a difficult task. While SQL Server 2005 introduced tools to help you understand how TempDB is utilized, many DBAs do not realize how certain optimizations in TempDB change the way to performance-tune its use.

## Objects

Before you begin to optimize TempDB in your environment, it is important to realize how SQL Server uses TempDB. TempDB holds three types of objects:

1. **Internal Objects:** These are essentially hidden objects; their metadata cannot be viewed in catalog views like `sys.objects` or `sys.all_objects`. Updates to internal objects do not generate log records (nor do page allocation operations, unless they are the product of a sort), which makes these operations faster than similar operations on user objects. Internal objects are largely intermediate data used at runtime to process data in hashes, spools, and sorts; to store large object variables; and to process cursors. Each internal object will occupy at least one IAM page and eight data pages, for a total of at least nine pages.
2. **Version Stores:** Version stores are used to store row versions generated by transactions for features such as snapshot isolation, triggers, MARS (multiple active result sets), and online index builds. Version stores consist of append-only store units that are highly optimized for sequential inserts and random look-up. Like internal objects, version stores do not appear in catalog views and inserts do not generate log records. There are two types of version stores:
  - o The online index build version store is for row versions from tables that have online index build operations on them.
  - o The common version store is for row versions from all other tables in all databases.
3. **User Objects:** These are both user-defined tables and indexes, and system catalog tables and indexes. Operations on user objects are logged in line with operations in other databases when SIMPLE recovery mode is used, but there is an important exception: TempDB uses a logging optimization that avoids logging the resulting value of INSERT and UPDATE operations; this optimization primarily benefits operations on heap and LOB data, and effectively reduces the amount of I/O traffic on the log device used by TempDB.

This paper focuses primarily on internal objects and user objects.

## Optimizations

Now that you know what is stored in TempDB, it is important to consider some of the other optimizations made to TempDB in SQL Server 2005, including the following:

- **Instant file initialization:** SQL Server 2005 does not zero out the NTFS file when the size of a file is increased, which minimizes overhead significantly when TempDB needs to auto grow. Without this feature, auto grow could take a long time and lead to application timeout. This optimization requires the following:
  - o Microsoft® Windows XP or Windows 2003 or higher must be used.
  - o The SQL Server (MSSQLSERVER) service account must be granted `SE_MANAGE_VOLUME_NAME`. Members of the Windows Administrator group have this right and can grant it to other users by adding them to the Perform Volume Maintenance Tasks security policy.
- **Optimized page allocation:** UP type page latches are used less when allocating pages and extents, and allocating the first eight pages of a table is more efficient. As a result there should be less contention during high concurrency periods.

- **Proportional fill:** When UP latches are used and multiple files are defined, SQL Server fills each file proportionally across files. This resolves a point of contention in SQL Server 2000.
- Temporary object and worktable caching and deferred drop: Cached query execution plan work tables are truncated, and the first nine pages are kept between plan executions. Table-valued functions, table variables, and local temporary tables are cached when they are used in compiled code (stored procedures, functions, or triggers), so subsequent creates are very fast. In addition, if the temporary object size remains under 8 MB, one data and one IAM page remain cached. Temporary tables are not cached if there is an explicit DDL on the table after it is created, if a named constraint is used, or if the temporary object is larger than 8 MB. When dropping a temporary object, SQL Server 2005 introduced a feature called deferred drop, which assigns drops of temporary tables to a background task so applications do not have to wait.

**Note:** When the system is low on memory and memory pressure forces a plan out of the procedure cache, the related work table and/or temporary object will be dropped as well.

**Tip:** Freeing the procedure cache or dropping compiled code objects that reference temporary objects will remove these objects from TempDB.

## Configuration and Sizing

TempDB supports one data filegroup and one log filegroup. Adding multiple data files may help resolve performance problems due to I/O operations by avoiding latch contention on allocation pages (see UP latch type optimization); all files should be sized equally and striped across fast disks. The recommended solution is to configure as many files as there are [logical] CPUs configured for the instance. Too many files will increase the cost of file switching, require more IAM pages, and increase manageability overhead.

### Data File—TempDB Objects

To correctly size TempDB, the objects created in the database must be considered, including internal objects and user objects.

#### Internal Objects

Because applications cannot directly access internal objects, you should review the query plans generated by your application SQL and pay particular attention to the following operators:

- **Sort:** Requires TempDB space to sort the full rowset.
- **Hash match:** Uses two inputs, one to build a hash table and another to probe its contents. You should review the number of rows and the row size returned by the first input operator.
- **Spool:** Requires TempDB space to store the full input rowset.

To estimate the space required by each operator, look at the number of rows and the row size reported by the operator. To calculate the space required, multiply the number of rows (estimated or actual) by the estimated row size.

**Note:** The estimated number of rows and estimated row size are only as accurate as the statistics in your database! Before you attempt to size TempDB, you should first address index and statistics maintenance for your key application databases.

## User Objects

For user objects, pay attention to application operations performed on user-defined tables, global and local temporary tables, indexes, and table variables. Space calculations can be performed on these objects just as they would on any standard user table. Applications have explicit control over how much data is loaded into these objects, just as they have control over how SQL Server is accessed.

If an application in your environment uses server-side cursors, you need to determine if static or keyset cursors are being used. For static cursors, space is required for the whole result. For keyset cursors, the space required is the average key size multiplied by the number of rows in the result set.

When you create an index, you have an option to sort in TempDB. The sort requires roughly the same amount of space as the index itself and may require the creation of a mapping index. To calculate the size required to create the index, multiply the average key size by the number of rows in the index.

Service Broker, DBCC, XML, and LOB variable operations also affect the size of internal objects. Generally, since Service Broker (and other features and utilities that leverage Service Broker) uses about 1 MB per dialog, space usage should not be an issue; however, poor Service Broker application design (that is, if an application starts dialogs but neglects to close them) can lead to space concerns in TempDB. DBCC CHECK is an example of a maintenance operation that might contend with applications for TempDB space. It is a worthwhile exercise to leverage the option in DBCC CHECKDB to estimate the space required to run the command. Finally, review your application code to understand the size required by any LOB or XML variables or parameters because they all consume space in TempDB in SQL Server 2005.

**Note:** A commonly accepted best practice when sizing TempDB is to include an additional 20% over and above the estimated size required to account for periods of high concurrency. Auto grow should be enabled to allow for future data growth.

To gain a better grasp of how your application is using TempDB at runtime, use the DMV **sys.dm\_db\_file\_space\_usage** to return TempDB file space allocation information by object category. By leveraging the information above with regard to each object type, you can keep track of how your application is using TempDB and tune your applications in accordance with your findings.

## Log Operations

Estimating the log space for TempDB is similar to standard database log sizing efforts. Because many operations do not generate log records in TempDB, and because TempDB is designed to truncate itself, your main concern is sizing the log file adequately to avoid frequent log truncations. You can review and trend the following three performance counters to determine appropriate log file sizing:

- **Database: Trend Log File(s) Size (KB)** to determine the size of the log files associated with TempDB. Allowing your log files to auto grow can help determine the watermark size achieved during periods of high concurrency in your application environment.
- **Database: Trend Log File(s) Used (KB)** to determine how much of the log is used. Your log files may get quite large at certain points in your application workload, which can point to long-running transactions or high concurrency periods. Your log file will have to be able to accommodate these periods, so analyzing this metric is important.
- Trend **Free Space in tempdb (KB)** for SQL Server 2005 (and up) instances to understand when TempDB is on the verge of growing a file or running out of space. This can be a telling metric with regard to how application code is using TempDB.

## Troubleshooting TempDB Using Quest Tools

### Foglight Performance Analysis for SQL Server

Performance Analysis for SQL Server provides the following performance advisories to help troubleshoot TempDB configuration and sizing issues:

- **Buffer Latch Contention**—Determine whether significant buffer latch contention exists, and if so, show high buffer latch waits as an indication of I/O bottlenecks and hot pages. Since buffer latching is generally not directly related to I/O contention, this advisory can be sensitive to the amount of memory available to SQL Server.
- **Compiles / Recompiles**—Determine whether significant CPU contention exists, and, if so, show the transact SQL statements that were observed to be experiencing a high number of recompiles and consuming a large amount of CPU resources. Applications whose SQL code is frequently recompiled will usually benefit from:
  - Evaluating the purpose of the statements involved and separating data definition commands from data modification code
  - Addressing out-of-date index statistics
  - Replacing temporary tables with variables or other logic
- **Database File Configuration**—Determine whether significant disk I/O contention exists, and, if so, show how poorly configured database files can lead to increased latch contention within the database, which in turn can lead to resource bottlenecks and increased contention within applications. This advisory alerts DBAs to a number of poor database file configuration scenarios for databases suffering from latch contention, including the following:
  - Data and log files configured on the same drive
  - Fewer database files than available CPUs, with special emphasis on TempDB
  - Fewer database files than available disk I/O devices
- **Database File Growth**—Determine whether significant disk I/O contention exists, and, if so, show databases that have been observed to take extents too frequently. This advisory alerts DBAs to databases that have taken frequent extents over a configurable time frame. When SQL Server grows a database file, that file is more prone to corruption and fragmentation, and the process is extremely CPU and I/O intensive.
- **Inefficient Query Plans**—Determine whether significant CPU contention exists, and, if so, illustrate how inefficient query plans can consume excessive CPU resources. Existing database schemas, application requirements, user-enabled report wizards, and other conditions can force inefficient queries to be executed in a production environment; this advisory alerts DBAs to queries using HASH joins and SORT operations that are resulting high CPU and I/O consumption.
- **Inefficient or Missing Indexes**—Determine whether significant disk I/O contention exists, and, if so, illustrate how missing or inefficient indexes create disk I/O bottlenecks. DBAs are required to evaluate application SQL code and ensure that the statements executed are as efficient as possible, which generally entails creating indexes to most efficiently extract the data. But if the application SQL code changes to either access the tables differently or to select more or different columns from the target tables, the existing indexes may no longer apply. This advisory illustrates where SQL code is not effectively using existing indexes and where statements are using table scans to gather data.

- **Internal Cache Latch Contention**—Determine whether significant internal cache latch contention exists, and if so, identify where the majority of the contention exists. Internal cache latches can be used for a variety of things; possibly the most common case is contention on internal caches (not buffer pool pages), especially when using heaps, text, or both. If solving LOG and PAGELATCH\_UP contention does not help, internal cache latch contention can usually best be treated by partitioning data.
- **Log Waits**—Determine whether significant log waits were observed, and if so, show how a number of factors can contribute to SQL Server logging slowdowns.

Whenever one of these performance advisories is generated by Performance Analysis, reviewing the context-sensitive information displayed will pinpoint how you can configure TempDB for optimal performance. Once you have reviewed the information and implemented strategies to adjust your TempDB configuration, Performance Analysis will assess whether your strategy has been successful.

### Change Tracking

Performance Analysis for SQL Server can track changes in your SQL Server environment to determine the effectiveness of configuration adjustments to TempDB over time. In particular, you can monitor changes to statement execution plans over time. While the advisories will pinpoint resource-intensive statements and point you toward addressing the statements that are having the greatest impact in your environment, keeping tabs on changes to execution plans (specifically focusing on SORT and HASH operators) is a good proactive measure in addressing potential bottlenecks in TempDB.

### Spotlight on SQL Server Enterprise

A number of Database Management Dashboards and Reporting and Trending Views can you assist in verifying TempDB file configurations and in sizing TempDB appropriately.

### The SQL Server Drill-down Dashboards

Using the Databases Management Dashboard, you can quickly determine the health of TempDB.

Configuring and sizing TempDB requires that you understand a number of metrics provide on this dashboard, including:

- File Size Total / Used / Free
- Active Transactions and Transactions Rate
- Log Truncation Rate
- Number of Tables including Rows and Reserved MB
- Number of Indexes

You can also create custom counters to trend your applications' usage of TempDB:

TempDB: Free Space (MB):

```
select SUM([unallocated_extent_page_count])/128
from sys.dm_db_file_space_usage
```

TempDB: Internal Objects (MB):

```
select SUM([internal_object_reserved_page_count])/128
from sys.dm_db_file_space_usage
```

TempDB: User Objects (MB):

```
select SUM([user_object_reserved_page_count])/128
from sys.dm_db_file_space_usage
```

### TempDB: Version Store (MB):

```
select SUM([version_store_reserved_page_count])/128  
from sys.dm_db_file_space_usage
```

Using the Reporting and Trending Views, you can further understand the activity that has occurred within TempDB. The Database File IO Statistics view displays the top database files by overall I/O rate, read rate, and write wait. Correlating the information displayed in this view with the Database Growth Report will help you evaluate whether TempDB was frequently auto-growing its data files and therefore understand whether you've sized TempDB appropriately.



# Recommended Reading

---

Microsoft has published a number of excellent white papers and tech briefs that cover the topics discussed in this document. For more detail on any of these topics and for Microsoft's stance on troubleshooting these issues, we recommend you reference the following documents:

SQL Server 2000 I/O Basics

(<http://technet.microsoft.com/en-us/library/cc966500.aspx>)

SQL Server I/O Basics, Chapter 2

(<http://technet.microsoft.com/en-us/library/cc917726.aspx>)

Specifying Query Plans with Plan Forcing

([http://technet.microsoft.com/en-us/library/ms190727\(SQL.90\).aspx](http://technet.microsoft.com/en-us/library/ms190727(SQL.90).aspx))

Optimizing SQL Server CPU Performance

(<http://technet.microsoft.com/en-us/magazine/2007.10.sqlcpu.aspx>)

Performance Tuning: Waits and Queues ([http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/Performance\\_Tuning\\_Waits\\_Queues.doc](http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/Performance_Tuning_Waits_Queues.doc))

Statistics Used by the Query Optimizer (<http://blogs.msdn.com/sqlblog/archive/2006/09/19/whitepaper-statistics-used-by-the-query-optimizer-in-microsoft-sql-server-2005.aspx>)

Troubleshooting Performance Problems in SQL Server 2005

(<http://download.microsoft.com/download/1/3/4/134644fd-05ad-4ee8-8b5a-0aed1c18a31e/TShootPerfProbs.doc>)

Working with tempdb in SQL Server 2005 (<http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/WorkingWithTempDB.doc>)



## About Quest Software, Inc.

Now more than ever, organizations need to work smart and improve efficiency. Quest Software creates and supports smart systems management products—helping our customers solve everyday IT challenges faster and easier. Visit [www.quest.com](http://www.quest.com) for more information.

## Contacting Quest Software

PHONE 800.306.9329 (United States and Canada)

If you are located outside North America, you can find your local office information on our Web site.

E-MAIL [sales@quest.com](mailto:sales@quest.com)

MAIL Quest Software, Inc.  
World Headquarters  
5 Polaris Way  
Aliso Viejo, CA 92656  
USA

WEB SITE [www.quest.com](http://www.quest.com)

## Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract.

Quest Support provides around-the-clock coverage with SupportLink, our Web self-service. Visit SupportLink at <https://support.quest.com>.

SupportLink gives users of Quest Software products the ability to:

- Search Quest's online Knowledgebase
- Download the latest releases, documentation, and patches for Quest products
- Log support cases
- Manage existing support cases

View the Global Support Guide for a detailed explanation of support programs, online services, contact information, and policies and procedures.



5 Polaris Way, Aliso Viejo, CA 92656 | PHONE 800.306.9329 | WEB [www.quest.com](http://www.quest.com) | E-MAIL [sales@quest.com](mailto:sales@quest.com)  
If you are located outside North America, you can find local office information on our Web site.

© 2009 Quest Software, Inc.  
ALL RIGHTS RESERVED.

Quest Software is a registered trademark of Quest Software, Inc. in the U.S.A. and/or other countries. All other trademarks and registered trademarks are property of their respective owners.  
HOD-SQLServer-US-AG-20091223