

# Top 10 Tips for Optimizing SQL Server Performance

---

Written by

Kevin Kline

Technical Strategy Manager for SQL Server Solutions

Quest Software

# Contents

---

- Introduction ..... 2
- 10. What Problem Are We Trying to Solve ..... 3
  - What Are Baselineing and Benchmarking? ..... 3
  - What Baselineing Can't Do ..... 4
- 9. Performance Counters – How to Cut to the Chase ..... 6
  - Operational Monitoring ..... 6
  - Bottleneck Monitoring ..... 6
- 8. Why Changing Sp\_Configure Settings Probably Won't Help ..... 8
- 7. I Have a Bottleneck – What Do I Do Now? ..... 9
- 6. SQL Profiler Is Your Friend ..... 11
  - Start a Trace ..... 11
- 5. Zen and the Art of Negotiating with Your SAN Administrator ..... 15
- 4. The Horror of Cursors (and Other Bad T-SQL) ..... 17
- 3. Plan Reuse – Recycling for SQL ..... 19
- 2. The Mystery of the Buffer Cache ..... 21
- 1. The Tao of Indexes ..... 23
  - sys.dm\_db\_index\_operational\_stats ..... 23
  - sys.dm\_db\_index\_usage\_stats ..... 24
- Conclusion ..... 25
- About the Author ..... 26

# 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 detailed information on specific issues and even less information on how to apply that specific knowledge to your own environment.

In this white paper, I'll discuss the 10 things I think you should know about SQL Server performance. Each item is a nugget of practical knowledge that you can immediately apply to your environment.

# 10. 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 from your organization's hardware investment, you need to ensure the SQL or application workload running on the database servers is executing as quickly and efficiently as possible.

There are a number of business issues that relate to 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 in 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 available, e.g., performance counters, dynamic management views (DMVs), and others, but there is very little information on what to do with this data and how to interpret it. This paper describes 10 tips that will be useful in the trenches, allowing you to turn some of this data into actionable information.

## 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.

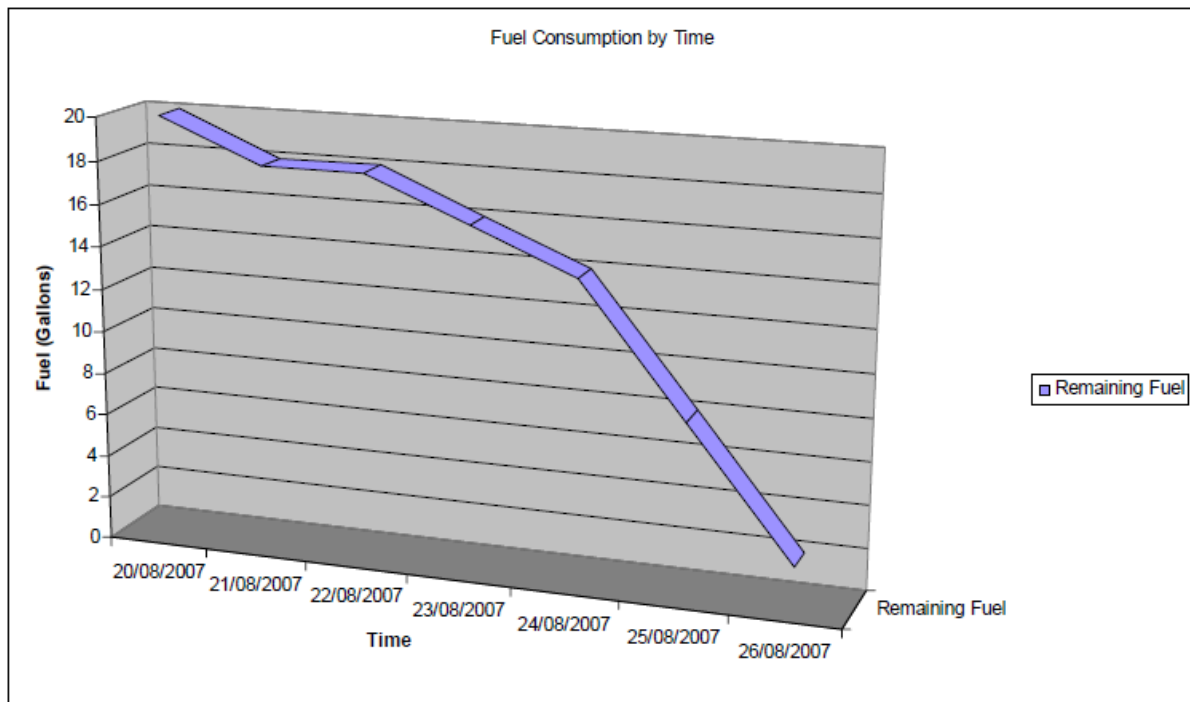
The practical upshot is that you can evaluate future behavior against a baseline to determine whether resource consumption has improved or worsened over time.

## 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 toward 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 your 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:

1. Increasing operational efficiency
2. Preventing 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 could also 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 the resource requirements you may 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	<p>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.</p> <p>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.</p>
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 of 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 the next section).

In SQL Server 2008 R2, if you identified a CPU bottleneck, the first thing that you would want to do is 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

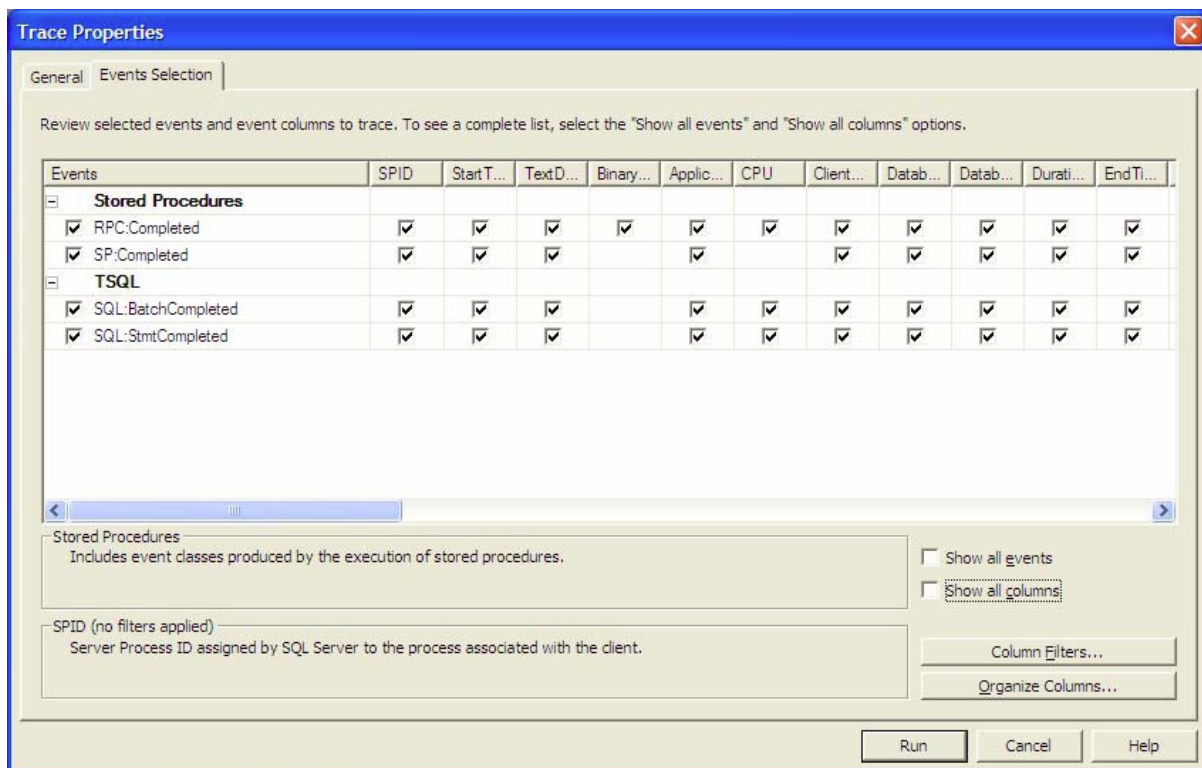
In the following step, I use Profiler Traces from a remote machine. There is nothing to stop you from using server side traces instead. The important part is what you do with the raw data once it gets into a database table.

### Start a Trace

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

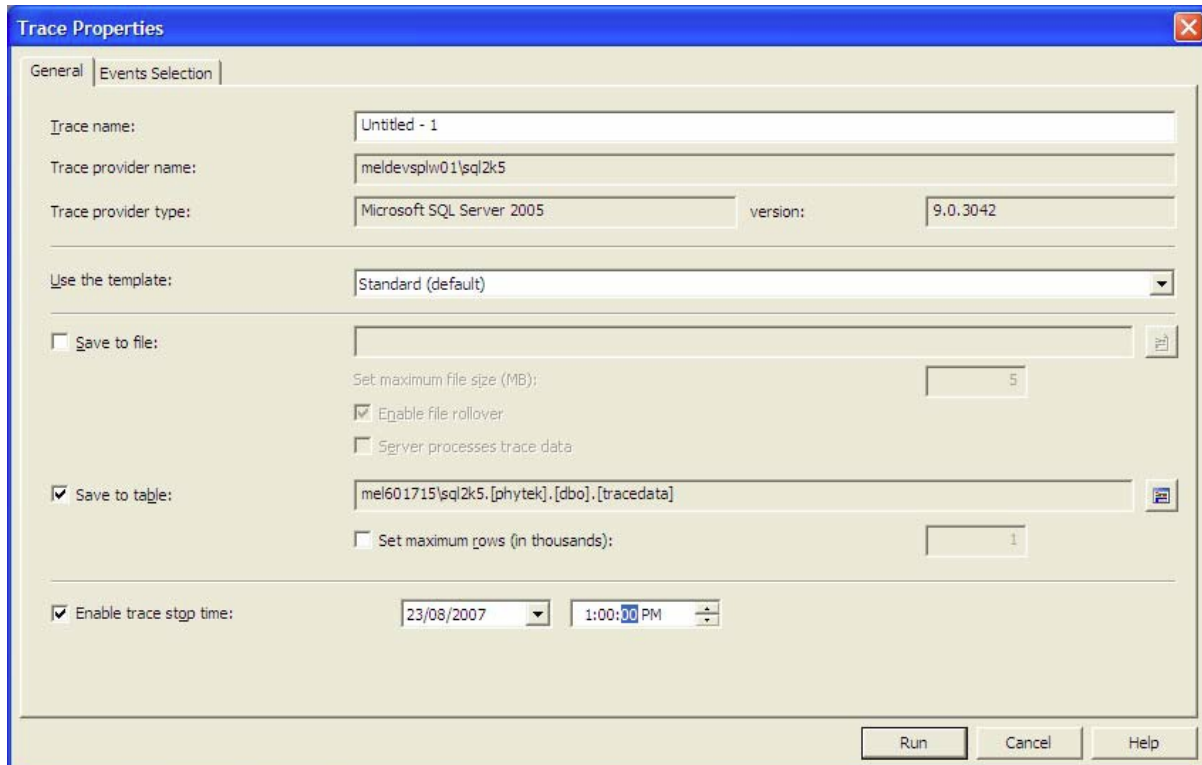
- RPC:Completed
- SP:Completed
- SQL:BatchCompleted
- SQL:StmtCompleted

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.]

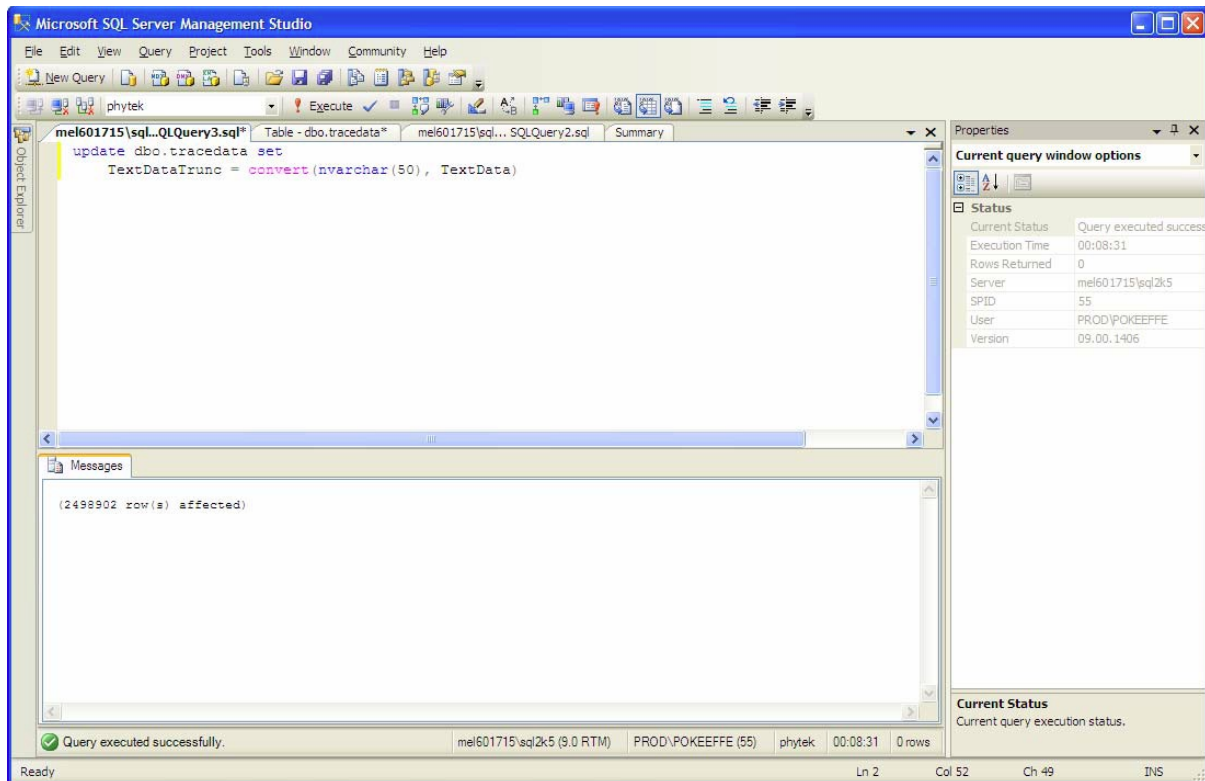
Once the trace is finished, the data should now be available in the database table that I configured. For those who wish to use server side tracing, we will also assume from this point that the trace data now exists in a table.

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 complete, I can 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:

The screenshot shows the Microsoft SQL Server Management Studio interface. The main window displays a SQL query that filters trace data for EventClass 12 (SQL:BatchCompleted) and summarizes it by TextDataTrunc. The query includes columns for TextDataTrunc, ExecCount, TotalCPU, and AvgCPU. The results pane shows 11 rows of data, with the first row being the most significant: 'use [loadtest]' with 269,705 executions, 136,217 total CPU, and 0 average CPU. The status bar at the bottom indicates the query executed successfully, returning 89,403 rows.

```

select
  TextDataTrunc,
  count(TextDataTrunc) as ExecCount,
  sum(cpu) as TotalCPU,
  avg(cpu) as AvgCPU
from
  dbo.tracedata
where
  EventClass = 12 --SQL:BatchCompleted
group by
  TextDataTrunc
order by
  ExecCount desc

```

TextDataTrunc	ExecCount	TotalCPU	AvgCPU
use [loadtest]	269705	136217	0
declare @Bat	294	342	1
decl	293	219	0
exec dbo.add_invoice_line_item @tran_id = 13248707	141	174	1
exec dbo.add_invoice_line_item @tran_id = 13248840	141	283	2
exec dbo.add_invoice_line_item @tran_id = 13254307	135	299	2
exec dbo.add_invoice_line_item @tran_id = 13253936	129	249	1
exec dbo.add_invoice_line_item @tran_id = 13253872	126	230	1
exec dbo.add_invoice_line_item @tran_id = 13254991	124	172	1
exec dbo.add_invoice_line_item @tran_id = 13254794	122	171	1
exec dbo.add_invoice_line_item @tran_id = 13248460	121	234	1

[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 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 `IoStallMS`, which is the amount of time users had to wait for I/O to complete on a file.



The screenshot shows a SQL query executed in Microsoft SQL Server Management Studio. The query is: `select db_name(dbId) as 'Database', * from ::fn_virtualfilestats(null,null) order by ioStallMS desc`. The results are displayed in a table with 13 columns and 16 rows. The columns are: Database, D..., Field, TimeStamp, NumberReads, BytesRead, ioStallReadMS, NumberWrites, BytesWritten, ioStallWriteMS, ioStallMS, BytesOnDisk, and FileHandle. The rows are sorted by ioStallMS in descending order.

Database	D...	Field	TimeStamp	NumberReads	BytesRead	ioStallReadMS	NumberWrites	BytesWritten	ioStallWriteMS	ioStallMS	BytesOnDisk	FileHandle
LoadTest	8	1	1218540796	83383	691798016	8212497	216326175	2149372026880	4078264547	4086477044	104098430976	0x0000076E
LoadTest	8	2	1218540796	411	20968960	2644	692200002	817976521216	1142130611	1142133255	5308416	0x0000089C
tempdb	2	2	1218540796	8	454656	0	114490	3865314816	498490	498490	786432	0x0000068C
tempdb	2	1	1218540796	2941	25395200	26328	19315	212525056	129801	156129	104857600	0x0000067E
tempdb	2	3	1218540796	2916	24035328	18002	4672	92577792	111053	129055	104857600	0x00000694
tempdb	2	5	1218540796	2936	24255512	22692	4584	91873280	98425	121117	104857600	0x00000684
tempdb	2	4	1218540796	2889	23797760	17637	4562	92037120	103385	121022	104857600	0x0000069C
master	1	2	1218540796	14	188416	717	205057	848560128	60171	60888	1835008	0x000005C4
QuestSoftware58	9	2	1218540796	11	417792	17	2801	9778176	31463	31480	2097152	0x00000520C
QuestWorkDatabase	13	1	1218540796	403	7045120	9917	4	32768	4	9921	5242880	0x0000088C
master	1	1	1218540796	139	3670016	6742	4386	44851200	2184	8926	4194304	0x0000050C
ReportServer\$SQL2K5	6	1	1218540796	176	4456448	7306	0	0	0	7306	4390912	0x000008A4
sqlnexus	12	1	1218540796	124	2547712	3042	1	8192	6	3048	66256896	0x000007FC
msdb	4	1	1218540796	254	5595136	2741	1	8192	2	2743	6291456	0x000008AC
model	3	1	1218540796	181	8585216	2278	4	32768	3	2281	2293760	0x0000063E
sqlnexus_snap	11	1	1218540796	216	1769472	1832	0	0	0	1832	1376256	0x000007E4

[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?

## 4. 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
```

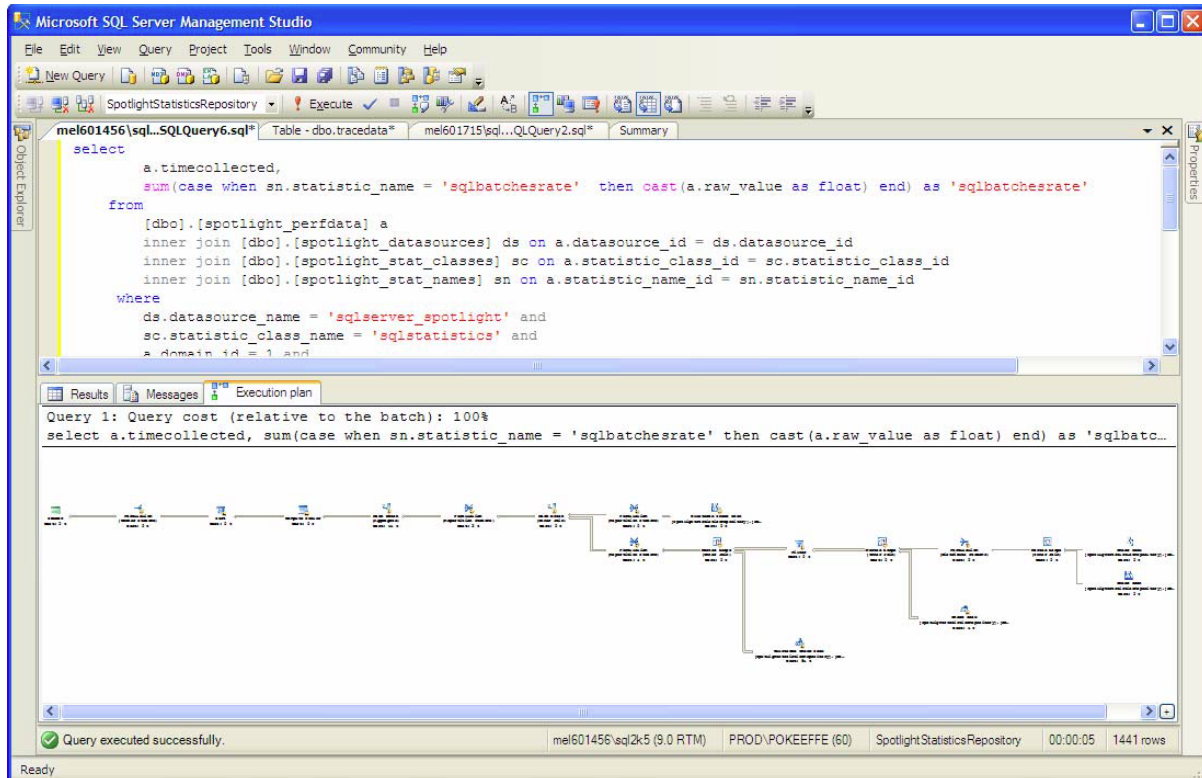
This is real code in a real production system. It can actually be reduced to:

```
UPDATE Patient SET ConfirmFlag = 'N'
```

```
WHERE PolicyGUID = @PolicyGUID
```

This refactored code will, of course, 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:

- "Professional SQL Server Execution Plan Tuning" by Grant Fritchey
- "SQL Server Internals & Troubleshooting" (paperback) by Christian Bolton, Brent Ozar, Justin Langford, James Rowland-Jones, Jonathan Kehayias, Cindy Gross, and Steven Wort

## 3. 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.

(Batch Requests/sec – SQL Compilations/sec) / 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. Therefore, you 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 `sys.dm_os_buffer_descriptors` DMV on SQL Server 2008 R2 (but available from 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 2008 R2 (though it works on SQL Server 2005 and later):

```

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.

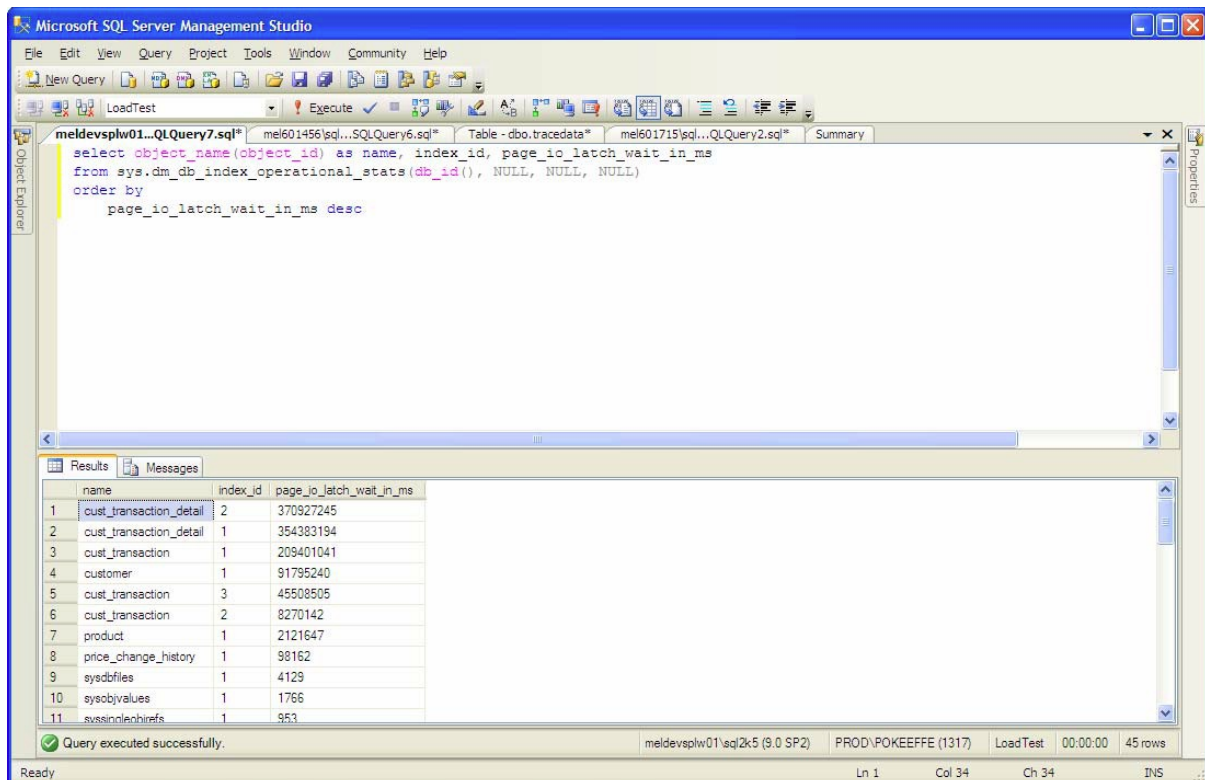
# 1. The Tao of Indexes

SQL Server 2008 R2 gives us some very useful new data on indexes, and you can get this data using DMVs starting at version SQL Server 2005.

## 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.



The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL query:

```
select object_name(object_id) as name, index_id, page_io_latch_wait_in_ms
from sys.dm_db_index_operational_stats(db_id(), NULL, NULL, NULL)
order by
    page_io_latch_wait_in_ms desc
```

The Results pane displays the following data:

	name	index_id	page_io_latch_wait_in_ms
1	cust_transaction_detail	2	370927245
2	cust_transaction_detail	1	354383194
3	cust_transaction	1	209401041
4	customer	1	91795240
5	cust_transaction	3	45508505
6	cust_transaction	2	8270142
7	product	1	2121647
8	price_change_history	1	98162
9	sysdfiles	1	4129
10	sysobjvalues	1	1766
11	sysinforefrefs	1	953

The status bar at the bottom indicates: Query executed successfully. meldevsplw01\sql2k5 (9.0 SP2) PROD\POKEEFEE (1317) LoadTest 00:00:00 45 rows

[Figure 8.]

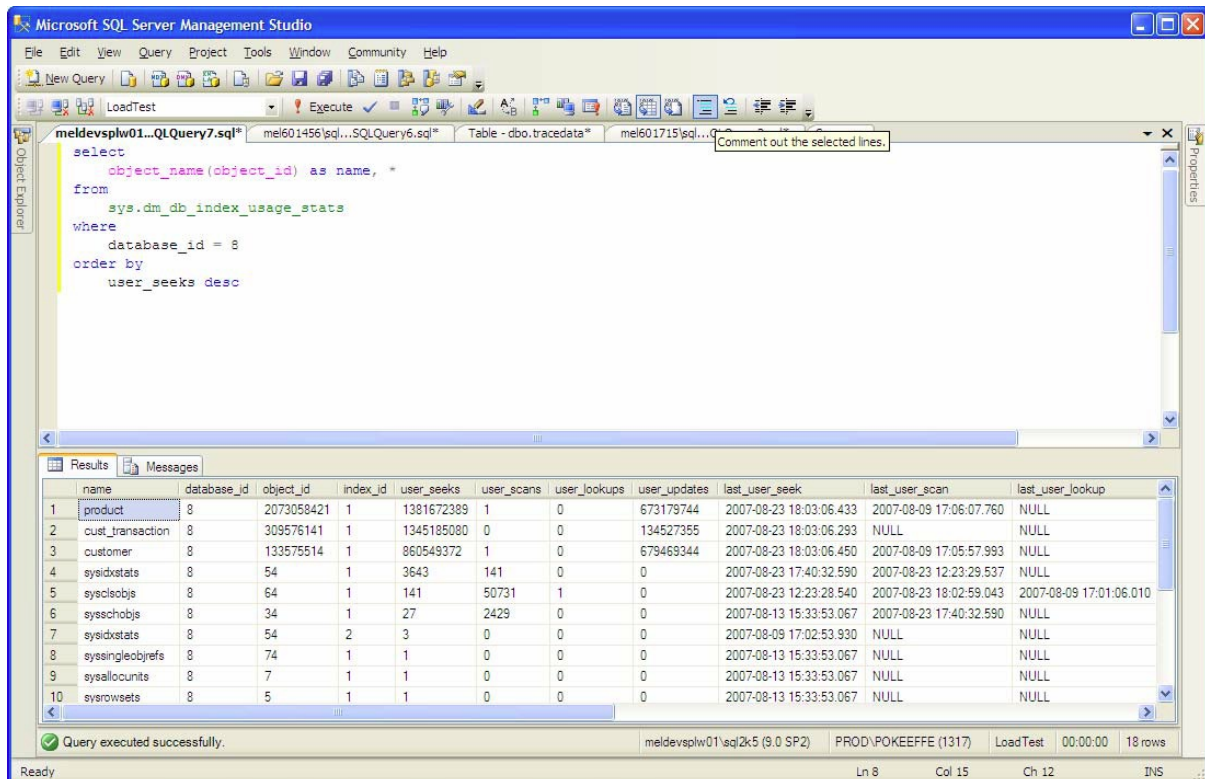
Figure 8 illustrates the output of a query that lists indexes by the total PAGE\_IO\_LATCH wait. This is very useful when trying to determine which indexes are involved in I/O bottlenecks.



## 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.



The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL code:

```
select
  object_name(object_id) as name, *
from
  sys.dm_db_index_usage_stats
where
  database_id = 8
order by
  user_seeks desc
```

The Results pane displays the following table:

	name	database_id	object_id	index_id	user_seeks	user_scans	user_lookups	user_updates	last_user_seek	last_user_scan	last_user_lookup
1	product	8	2073058421	1	1381672389	1	0	673179744	2007-08-23 18:03:06.433	2007-08-09 17:06:07.760	NULL
2	cust_transaction	8	309576141	1	1345185080	0	0	134527355	2007-08-23 18:03:06.293	NULL	NULL
3	customer	8	133575514	1	860549372	1	0	679469344	2007-08-23 18:03:06.450	2007-08-09 17:05:57.993	NULL
4	sysidxstats	8	54	1	3643	141	0	0	2007-08-23 17:40:32.590	2007-08-23 12:23:29.537	NULL
5	sysclsobje	8	64	1	141	50731	1	0	2007-08-23 12:23:28.540	2007-08-23 18:02:59.043	2007-08-09 17:01:06.010
6	syschobjs	8	34	1	27	2429	0	0	2007-08-13 15:33:53.067	2007-08-23 17:40:32.590	NULL
7	sysidxstats	8	54	2	3	0	0	0	2007-08-09 17:02:53.930	NULL	NULL
8	sysinglecbjrefe	8	74	1	1	0	0	0	2007-08-13 15:33:53.067	NULL	NULL
9	sysallocunits	8	7	1	1	0	0	0	2007-08-13 15:33:53.067	NULL	NULL
10	sysrowsets	8	5	1	1	0	0	0	2007-08-13 15:33:53.067	NULL	NULL

The status bar at the bottom indicates: Query executed successfully. meldevsplw01\sql2k5 (9.0 SP2) PROD\POKEEFEE (1317) LoadTest 00:00:00 18 rows

[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 and newer versions, you can.

# Conclusion

---

On reflection, there are far more than 10 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. So, remember these 10 things when optimizing SQL Server performance:

10. Benchmarking facilitates comparisons of workload behavior and lets you spot abnormal behavior because you have a good indication of what normal behavior is.
9. Performance Counters give you quick and useful information about currently running operations.
8. Changing server settings usually yields limited returns.
7. DMVs help you identify performance bottlenecks quickly.
6. Learn to use SQL Profiler and traces.
5. SANs are more than just I/O.
4. Cursors and other bad T-SQL frequently return to haunt applications.
3. Maximize plan reuse for better SQL Server caching.
2. Learn how to read the SQL Server buffer cache and how to minimize cache thrashing.

And the number one tip for optimizing SQL Server performance:

1. Master indexing by learning how indexes are used and how to counteract the characteristics of bad indexes.

# About the Author

---

Kevin Kline is the Technical Strategy Manager for SQL Server Solutions at Quest Software, a leading provider of award-winning tools for database management and application monitoring. He is a founding board member and former president of the international Professional Association for SQL Server (PASS) and frequently contributes to database technology magazines, web sites, and discussion forums. Kevin also serves the community as an adviser to the SQL Saturday education program as well as a curriculum adviser for both the University of Washington and Purdue University at Calumet in their IT and Computer Science departments.

Kevin's most popular book is "SQL in a Nutshell" (now in its third edition) published by O'Reilly Media. Kevin is also author or co-author on seven other IT books, including "Transact-SQL Programming," "Database Benchmarking: A Practical Approach," and "Professional SQL Server 2008 Relational Database Design and Optimization."

A top-rated speaker, he appears at international conferences like Microsoft TechEd, DevTeach, PASS, Microsoft IT Forum, SQL Connections, and the Best Practices Conference.

Beginning his career as a lowly hardware jockey working with PCs, Digital VAX, and Intergraph Unix workstations, Kevin has worked on multiple large-scale database projects throughout his career at Deloitte & Touche, NASA, and the U.S. Army.

When Kevin isn't working on technology issues, he enjoys spending time with his wife, Rachel, his four kids, his three stepchildren, and his basset hound and ginger kitty.

His online presences include:

- Blog: <http://kevinekline.com>
- Twitter: <http://twitter.com/kekline>

© 2010 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, SQLLab, 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—April, 2011

## About Quest Software, Inc.

Quest Software (Nasdaq: QSFT) simplifies and reduces the cost of managing IT for more than 100,000 customers worldwide. Our innovative solutions make solving the toughest IT management problems easier, enabling customers to save time and money across physical, virtual and cloud environments. For more information about Quest solutions for application management, database management, Windows management, virtualization management and IT management, go to [www.quest.com](http://www.quest.com).

## 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.

EMAIL [sales@quest.com](mailto:sales@quest.com)

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

## 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) | EMAIL [sales@quest.com](mailto:sales@quest.com)

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

© 2011 Quest Software, Inc.  
ALL RIGHTS RESERVED.

Quest, Quest Software, the Quest Software logo [and product name] are registered trademarks 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. WPD-Optimize-SQL-ServerPerf-US-KS-20110407