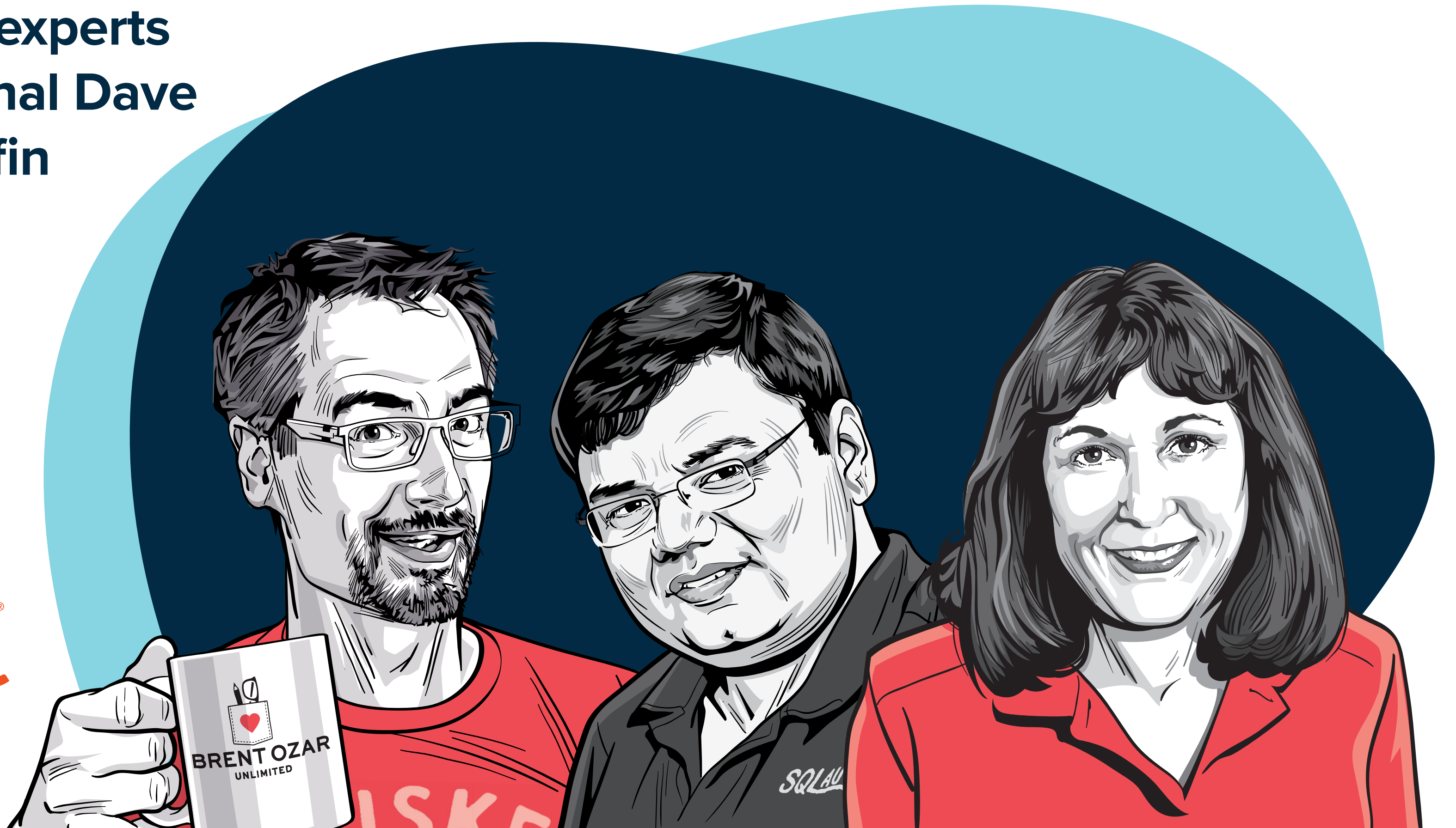
AN EXPERT GUIDE TO SQL SERVER PERFORMANCE TUNING

Top database performance tuning tips and tricks from industry experts Brent Ozar, Pinal Dave and Janis Griffin



Quest®

The Trials and Tribulations of SQL Server Performance Tuning

Database professionals agree – SQL Server performance tuning is hard. And on top of that, it never stops because complex database environments are always changing with upgrades, application updates and queries. It often feels like as soon as you get one query optimized, there's another one right behind it that's eating CPU time or clogging memory or otherwise slowing down the entire database. Then, add to that, the instances when the latest SQL Server version itself has made performance worse instead of making it better as promised.

As with many daunting tasks, often the toughest part of performance tuning in SQL Server is knowing where to start. So, we asked several SQL Server experts to share their advice on what to look for when the database is performing slowly. During Quest's Database Training Days webcast series, we invited Pinal Dave, SQL Server Performance Tuning Expert, Janis Griffin, Oracle ACE Director and Brent Ozar, Microsoft Certified Master to host hands-on training sessions where they identified quick wins, common performance inhibitors and the best places to begin and focus your efforts when tuning SQL Server. We've compiled their top tips to help guide you in your performance tuning efforts.

Pinal Dave's Kick Start Advice

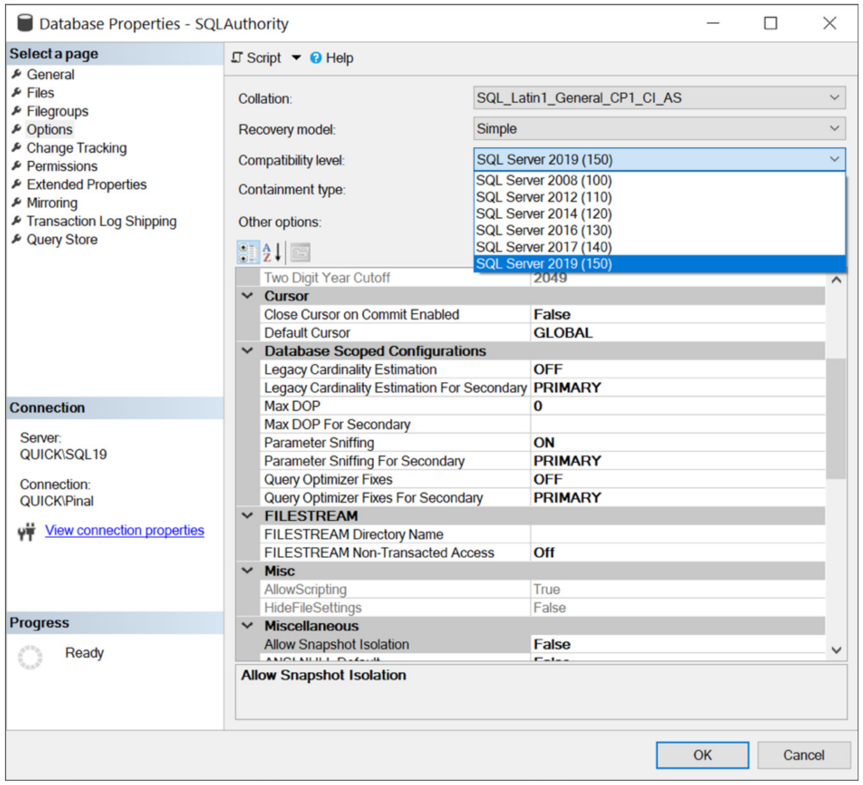
"99% of the time, when you change compatibility level, you get better performance."

Pinal Dave has tuned many SQL Server databases, and one of the most common complaints he hears is that SQL Server is actually running more slowly after an upgrade than it did before. This is particularly frustrating since a software upgrade may also include an investment in hardware that should be yielding positive performance results. So, what does Pinal suggest you do?

CHECK THE COMPATIBILITY LEVEL

In the case of a SQL Server upgrade that doesn't behave as expected, the first thing to check is the system's compatibility level. Compatibility level is a database configuration that determines which algorithm SQL Server uses to execute queries and support available features. After an upgrade, you may be running the latest version of SQL Server but the compatibility level instructs it to behave and perform like it's still an earlier version.

Figure 1 is a screenshot of SQL Server 2019 showing the different compatibility levels available in the SQL Server Database Properties:



Database Properties - SQLAuthority

Select a page
- General
- Files
- Filegroups
- Options
- Change Tracking
- Permissions
- Extended Properties
- Mirroring
- Transaction Log Shipping
- Query Store

Connection
Server:
QUICK\SQL19
Connection:
QUICK\Pinal
View connection properties

Progress
Ready

Script Help

Collation:	SQL_Latin1_General_CP1_CI_AS
Recovery model:	Simple
Compatibility level:	SQL Server 2019 (150)
Containment type:	
Other options:	

SQL Server 2008 (100)
SQL Server 2012 (110)
SQL Server 2014 (120)
SQL Server 2016 (130)
SQL Server 2017 (140)
SQL Server 2019 (150)

Two Digit Year Cutoff	2049
Cursor	
Close Cursor on Commit Enabled	False
Default Cursor	GLOBAL
Database Scoped Configurations	
Legacy Cardinality Estimation	OFF
Legacy Cardinality Estimation For Secondary	PRIMARY
Max DOP	0
Max DOP For Secondary	
Parameter Sniffing	ON
Parameter Sniffing For Secondary	PRIMARY
Query Optimizer Fixes	OFF
Query Optimizer Fixes For Secondary	PRIMARY
FILESTREAM	
FILESTREAM Directory Name	
FILESTREAM Non-Transacted Access	Off
Misc	
AllowScripting	True
HideFileSettings	False
Miscellaneous	
Allow Snapshot Isolation	False

Allow Snapshot Isolation

OK Cancel

Figure 1: Database Properties dialog box in SQL Server 2019

Quest

Why else would you want to change the compatibility level? In addition to updating the underlying logic by which SQL Server operates, the latest compatibility level also allows SQL Server to use new features and built-in optimizations. As an example, in 2017, SQL Server introduced adaptive join, where it dynamically decides which type of join will work most efficiently for a query. An earlier compatibility level would prevent the system from using that feature.

UNDERSTAND THE CARDINALITY ESTIMATION

When determining the best query execution plan, the Query Optimizer relies heavily on cardinality estimation – or the number of rows that will be processed at each level of the plan. The cardinality estimation is a significant contributor to the overall estimation of the plan cost – so improved cardinality leads to better estimated costs and faster plan execution.

The cardinality estimator has become more accurate in later versions of SQL Server, and most queries benefit from running the latest version. But there are times when queries won't perform as well with the latest cardinality estimator. You can test this by running the query using the earlier and later versions of the cardinality estimator, and clearing your query store in between. The on/off toggle for Legacy Cardinality Estimation is found in the Database Properties area, and is visible in Figure 1.

Again, the goal is to have the best cardinality estimation for your compatibility level, so your queries execute at a lower cost.

PAY ATTENTION TO ROW GOALS

In SQL Server, when the Query Optimizer estimates the cost of a query execution plan, it assumes that all qualifying rows from all sources must be processed and the entire results set returned at once. However, certain query keywords (such as TOP, FAST, IN, etc.) instruct the Query Optimizer to build an execution plan that returns a smaller number of rows faster – these are row goals that regulate how the results set is returned.

According to Microsoft, "If the row goal plan is applied, the estimated number of rows in the query execution plan is reduced. This is because the plan assumes that a smaller number of rows will have to be processed in order to reach the row goal."

Most of the time, a row goal strategy works as expected. Other times, it can go very wrong because many more rows need to be processed than the Optimizer estimated. When searching for the root cause of the resulting poor performance, you want to know whether the divergence you see in estimated versus actual rows is caused by row goal logic.

Prior to SQL Server 2019, there was no way to see row goals in execution plans, making them hard to detect when tuning queries. But 2019 introduced a visible EstimateRowsWithoutRowGoal attribute to each plan operator affected by a row goal. With this information, you can determine whether the row goals logic is the cause of the performance problem.

Janis Griffin's Path to Optimization

"Have you ever found yourself tuning something and no one notices?"

MONITOR WAIT TIME

Many database professionals undertaking SQL Server performance tuning focus on resource utilization metrics, such as response time. This is an important part of the equation that tells you whether your hardware is up to the task of processing the query, but it's also critical to know what's happening when the query isn't being processed. What is it waiting for? That's the concept behind wait time analysis.

SQL Server allows you to monitor both the total wait time, as well as elapsed time for each step of the query. Wait types offer invaluable clues about the amount of time a query is taking and the resources it's consuming as it runs. Therefore, it's useful to know how to interpret the different wait types at work in your database, including locking/blocking, I/O problems, latch contention and network slowdown.

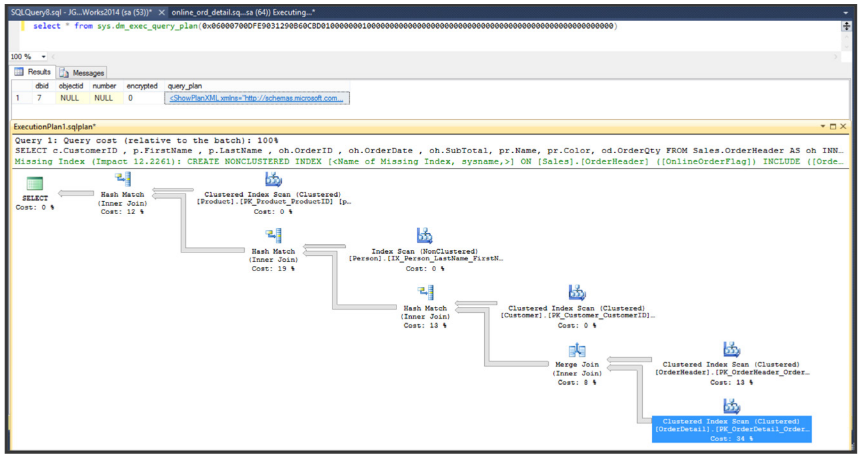
Wait time analysis must be done by viewing the wait types over time, otherwise, you're just seeing a snapshot of what's happening. There are benefits to capturing wait time information over a longer period. First, you're collecting valuable baseline metrics that enable you to set an acceptable performance improvement goal and stop when you reach it. Also, when you're armed with wait time information, you can identify the biggest contributor to slow performance and focus on fixing it.

REVIEW THE EXECUTION PLAN

Possibly the most important tool in your performance tuning arsenal is the execution plan, so it is vital that you know how to read and interpret what you see in the plan. The execution plan gives you a view into what's being done (filtering, sorting, joining) to the objects (tables and indexes) being used by the query.

An estimated execution plan is SQL Server's best guess at how the query will perform when running and shows the anticipated cost (in CPU time and I/O) of executing the query. An execution plan is read from right to left and bottom to top, with the thickness of the connecting arrows indicating the number of rows being passed in the step.

When you examine the estimated or actual execution plan, you'll want to look at CPU/IO costs and row counts, as well as review the predicate information to see how parameters are being interpreted. You'll also want to review join methods as you look for the most expensive steps in the plan. SQL Server provides a wealth of information in execution plan icon Tool Tips and the drill downs into the operator properties.



SQLQuery8.sql - JG...Works2014 (sa (53))* online_ord_detail.sq...sa (64)) Executing...*

```
select * from sys.dm_exec_query_plan(0x06000700DFE9031290B60CBD01000000010000000000000000000000000000000000000000000000000000)
```

	dbid	objectid	number	encrypted	query_plan
1	7	NULL	NULL	0	<ShowPlanXML xmlns="http://schemas.microsoft.com...

ExecutionPlan1.sqlplan*

```
Query 1: Query cost (relative to the batch): 100%
SELECT c.CustomerID , p.FirstName , p.LastName , oh.OrderID , oh.OrderDate , oh.SubTotal, pr.Name, pr.Color, od.OrderQty FROM Sales.OrderHeader AS oh INN...
Missing Index (Impact 12.2261): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [Sales].[OrderHeader] ([OnlineOrderFlag]) INCLUDE ([Orde...
```

100 %
Results
Messages
SELECT Cost: 0 %
Hash Match (Inner Join) Cost: 12 %
Clustered Index Scan (Clustered) [Product].[PK_Product_ProductID] [p... Cost: 0 %
Hash Match (Inner Join) Cost: 19 %
Index Scan (NonClustered) [Person].[IX_Person_LastName_FirstN... Cost: 0 %
Hash Match (Inner Join) Cost: 13 %
Clustered Index Scan (Clustered) [Customer].[PK_Customer_CustomerID]... Cost: 0 %
Merge Join (Inner Join) Cost: 8 %
Clustered Index Scan (Clustered) [OrderHeader].[PK_OrderHeader_Order... Cost: 13 %
Clustered Index Scan (Clustered) [OrderDetail].[PK_OrderDetail_Order... Cost: 34 %

Figure 2: Read an execution plan from right to left and from bottom to top

GATHER OBJECT INFORMATION AND FIND THE DRIVING TABLE

Once you've identified those expensive steps, you'll want to dig deeper into the objects underlying them. Get the table definition, review the column information and make note of the row counts. You're looking for performance degraders that eat up memory. Be sure to examine the WHERE and JOIN clauses, review indexes and index definitions, existing keys and constraints, and look for implicit conversions.

Table row counts are important as you find the driving table – this is the table that will return the least amount of data and minimize the number of logical reads. Proper filtering done early on this table will further reduce the amount of data that needs to be read in later steps. The table with the most selective filter is going to be your best choice for the driving table. A technique called SQL diagramming can be used to map this out.

Brent Ozar Breaks it All Down

"Success means a performance boost that end users notice."

Brent Ozar is one of the determined few who attained the status of Microsoft Certified Master in 2011 and is highly regarded as a SQL Server expert, trainer and consultant. Brent helps clients understand how to measure their SQL Server performance, since tuning makes no sense unless you know whether your performance metrics are actually changing.

SELECT AND MEASURE THE RIGHT METRICS

We touched on baseline metrics earlier, and for SQL Server performance tuning you should know generally-accepted rules of thumb as well as what's normal for your particular system environment. The amount of data you have will guide your resource procurement decisions, along with which version of SQL Server can handle your active data. So, first understand how much data you're dealing with.

With the right hardware and software in place, you'll want to understand how busy the server is by examining the data point batch requests per second. It's essentially a performance counter – how many T-SQL command batches are received by the server each second. It will vary over time, perhaps seasonally or cyclically, depending on what's happening on the server. It's a first place to look when you hear that queries are running slowly – it's possible they're running slowly because a lot of them are occurring at the same time. A dramatic spike in batch requests/second will be telling, but you have to know your system's baseline for comparison.

The next important metric to keep as a baseline is how hard the server is working to deliver query results. This is encapsulated by a wait time ratio, which is the relationship between the CPU time and total database time needed to complete a process. It's an effective indicator of how long tasks are waiting in the queue for available resources before being processed. Ideally, the wait time ratio will be 100%, but 90% efficiency is good. If your wait time ratio drops below what's generally normal for your system, begin your investigation by analyzing the queries that ran during that time period.

UNDERSTAND INDEX FRAGMENTATION AND FILL FACTOR IMPACTS

Many database professionals look at index fragmentation statistics as a performance measure, when in fact, they're not useful. In a new or rebuilt index, the data "pages" are all full and stored in order. But, as more data gets added, pages become split: not all pages are full and they occur out of order. This is the vital difference between external and internal fragmentation:

- External fragmentation – refers to pages being out of order
- Internal fragmentation – refers to the empty space on a page

External fragmentation has little impact on the speed of maintenance tasks, running queries in RAM or reading data from disk. But trying to fix external fragmentation by reorganizing and rebuilding indexes can actually make performance worse by causing internal fragmentation. Internal fragmentation is controlled by the fill factor, and any setting under 100 tells SQL Server to leave free space on index pages for new records to be added.

If an index is rebuilt with a reduced fill factor, page count gets higher. With more pages in the index, more reads are needed from disk into memory and overall performance will slow down.

“Whatever your top wait stat is, that’s where you focus your performance tuning”

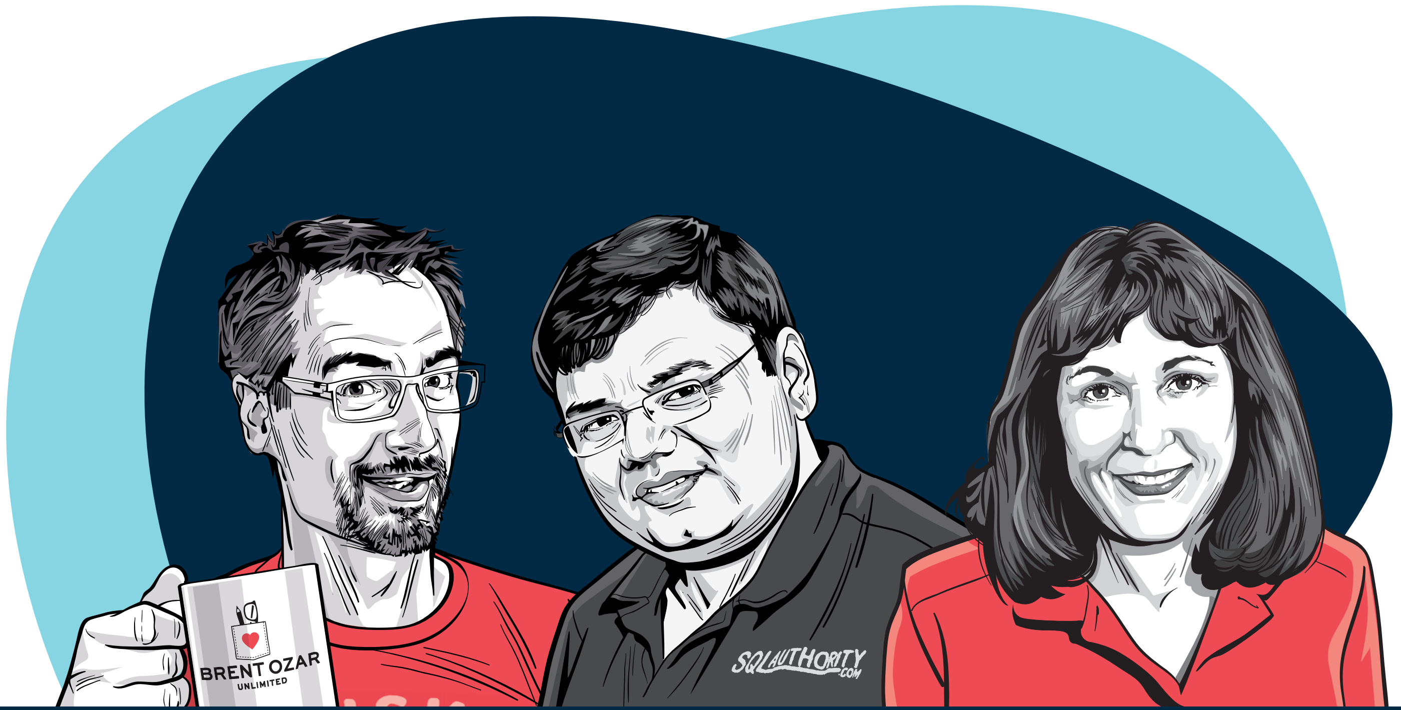
DID WE MENTION WAIT TIME?

Wait time is such an important element of performance tuning, yet SQL Server doesn’t offer an easy way to monitor it over time. You need to collect the results, then interpret them to determine where query tasks are spending their time or getting hung up. The area where a query is spending the most time should be the first target of your performance tuning efforts.

Common wait types include:

Wait Type	Explanation	What to Do
CXPACKET	Indicates there are parallel query plans running	Investigate where they are occurring and determine whether the parallelism is valid. If valid, is it working normally or is skewed parallelism happening?
PAGEIOLATCH	Query is waiting to read pages from disk	Long waits may indicate problems with the disk subsystem, but also think about why SQL Server is doing so many reads. Possibly tune the query’s indexes.
ASYNC_NETWORK_IO	Occurs on network writes when the task is blocked behind the network	Not much can be done. Could be due to network latency or shared VM.
LCK*	Occurs when a task is waiting to acquire an Intent Exclusive (IX) lock on a resource	Many ways to fix this, but ultimately you need to identify the thread that’s holding the lock and blocking everything else.
WRITELOG	Occurs while waiting for a log flush to complete.	Reduce how much transaction log is generated and how often log flushes are occurring.
RESOURCE_ SEMAPHORE	Happens when a query is waiting for an execution memory grant so it can begin executing operations like sorts and hashes.	Identify and then tune queries that need incorrectly large memory grants, or troubleshoot general memory shortages on the server.

Advanced performance monitoring tools allow you to track wait type statistics over time. Increases in wait time may be due to higher batch requests/second, poorly-tuned queries, slow storage or shared hardware. Conversely, decreases in wait time could mean there are fewer batch requests/second, well-tuned queries and indexes, and more available memory.



Watch the entire Database Training Days series on-demand now.

Further Insights Await

Quest's Database Training Days webcast series brought together industry experts who have made just about every SQL Server performance tuning mistake possible, and they have been kind enough to share that hard-earned wisdom in each of their interactive training sessions throughout the series. This e-book just scratches the surface on the tips and insights, detailed case studies and various query demonstrations candidly displayed. If you'd like to explore these concepts further, you can watch the entire Database Training Days series on-demand.

Quest

ABOUT QUEST

Quest provides software solutions for the rapidly changing world of enterprise IT. We help simplify the challenges caused by data explosion, cloud expansion, hybrid data centers, security threats and regulatory requirements. We're a global provider to 130,000 companies across 100 countries, including 95% of the Fortune 500 and 90% of the Global 1000. Since 1987, we've built a portfolio of solutions which now includes database management, data protection, identity and access management, Microsoft platform management and unified endpoint management. With Quest, organizations spend less time on IT administration and more time on business innovation. For more information, visit www.quest.com.

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

Quest Software Inc.
Attn: LEGAL Dept
4 Polaris Way
Aliso Viejo, CA 92656

Refer to our website (www.quest.com) for regional and international office information.

EBook-DatabaseTrainingDays-US-KA-59463