Measure CPU Behavior in Linux

You can still use `sys.dm_os_wait_stats` when running on Linux. This will give you wait statistics that can indicate a CPU load. Otherwise, you'll need to go the Linux system itself. The generally recommended method for looking at CPU is to use the `top` command. The output from that tool is documented here: `https://bit.ly/2KbZmuZ`.

Processor Bottleneck Resolutions

A few of the common processor bottleneck resolutions are as follows:

- Optimizing application workload
- Eliminating or reducing excessive compiles/recompiles
- Using more or faster processors
- Not running unnecessary software

Let's consider each of these resolutions in turn.

Optimizing Application Workload

To identify the processor-intensive queries, capture all the SQL queries using Extended Events sessions (which I will discuss in the next chapter) and then group the output on the CPU column. Another method is to take advantage of the Query Store (discussed in Chapter 11). You can retrieve information from `sys.query_store_runtime_stats` to see multiple, aggregated, CPU metrics on a per-query basis. The queries with the highest amount of CPU time contribute the most to the CPU stress. You should then analyze and optimize those queries to reduce stress on the CPU. Frequently, the cause for CPU stress is not extensive calculations within the queries but actually contention within logical I/O. Addressing I/O issues can often help you resolve CPU issues as well. You can also query directly against the `sys.dm_exec_query_stats` or `sys.dm_exec_procedure_stats` `dynamic management` view to see immediate issues in real time. Finally, using both a query hash and a query plan hash, you can identify and tune common queries or common execution plans (this is discussed in detail in Chapter 14). Most of the rest of the chapters in this book are concerned with optimizing application workload.

Eliminating Excessive Compiles/Recompiles

A certain number of query compiles and recompiles is simply to be expected, especially, as already noted, when working with ORM tools. It's when there is a large number of these over-sustained periods that a problem exists. It's also worth noting the ratio between them. Having a high number of compiles and a low number of recompiles means that few queries are being reused within the system (query reuse is covered in detail in Chapter 9). A high number of recompiles will cause high processor use. Methods for addressing recompiles are covered in Chapter 17.

Using More or Faster Processors

One of the easiest resolutions, and one that you will adopt most of the time, is to increase system processing power. However, because of the high cost involved in a processor upgrade, you should first optimize CPU-intensive operations as much as possible.

The system's processing power can be increased by increasing the power of individual processors or by adding more processors. When you have a high % Processor Time counter and a low Processor Queue Length counter, it makes sense to increase the power of individual processors. In the case of both a high % Processor Time counter and a high Processor Queue Length counter, you should consider adding more processors. Increasing the number of processors allows the system to execute more requests simultaneously.

Not Running Unnecessary Software

Corporate policy frequently requires virus checking software be installed on the server. You can also have other products running on the server. When possible, no unnecessary software should be running on the same server as SQL Server. Exterior applications that have nothing to do with maintaining the Windows Server or SQL Server are best placed on a different machine.

Network Bottleneck Analysis

In SQL Server OLTP production environments, you will find few performance issues that are because of problems with the network. Most of the network issues you face in an OLTP environment are in fact hardware or driver limitations or issues with switches or routers. Most of these issues can be best diagnosed with the Network Monitor tool. However, Performance Monitor also provides objects that collect data on network activity, as shown in Table 4-2.

Table 4-2. *Performance Monitor Counters to Analyze Network Pressure*

Object (Instance[,InstanceN])	**Counter**	**Description**	**Value**
Network Interface (Network card)	Bytes Total/sec	Rate at which bytes are transferred on the NIC	Average value < 50% of NIC capacity, but compare with baseline
Network Segment	% Net Utilization	Percentage of network bandwidth in use on a network segment	Average value < 80% of network bandwidth, but compare with baseline

Bytes Total/Sec

You can use the Bytes Total/sec counter to determine how the network interface card (NIC) or network adapter is performing. The Bytes Total/sec counter should report high values to indicate a large number of successful transmissions. Compare this value with that reported by the Network Interface\Current Bandwidth performance counter, which reflects each adapter's bandwidth.

To allow headroom for spikes in traffic, you should usually average no more than 50 percent of capacity. If this number is close to the capacity of the connection and if processor and memory use are moderate, then the connection may well be a problem.

% Net Utilization

The % Net Utilization counter represents the percentage of network bandwidth in use on a network segment. The threshold for this counter depends on the type of network. For Ethernet networks, for example, 30 percent is the recommended threshold when SQL Server is on a shared network hub. For SQL Server on a dedicated full-duplex network, even though near 100 percent usage of the network is acceptable, it is advantageous to keep the network utilization below an acceptable threshold to keep room for the spikes in the load.

Note You must install the Network Monitor Driver to collect performance data using the Network Segment object counters.

In Windows Server 2012 R2, you can install the Network Monitor Driver from the local area connection properties for the network adapter. The Network Monitor Driver is available in the network protocol list of network components for the network adapter.

You can also look at the wait statistics in `sys.dm_os_wait_stats` for network-related waits. But, one that frequently comes up is ASYNC_NETWORK_IO. While this can be an indication of network-related waits, it's much more common to reflect waits caused by poor programming code that is not consuming a result set efficiently.

Network Bottleneck Resolutions

A few of the common network bottleneck resolutions are as follows:

- Optimizing application workload
- Adding network adapters
- Moderating and avoiding interruptions

Let's consider these resolutions in more detail.

Optimizing Application Workload

To optimize network traffic between a database application and a database server, make the following design changes in the application:

- Instead of sending a long SQL string, create a stored procedure for the SQL query. Then, you just need to send over the network the name of the stored procedure and its parameters.
- Group multiple database requests into one stored procedure. Then, only one database request is required across the network for the set of SQL queries implemented in the stored procedure. This becomes extremely important when talking about Azure SQL Database.
- Request a small data set. Do not request table columns that are not used in the application logic.
- Move data-intensive business logic into the database as stored procedures or database triggers to reduce network round-trips.
- If data doesn't change frequently, try caching the information on the application instead of frequently calling the database for information that is going to be exactly the same as the last call.
- Minimize network calls, such as returning multiple result sets that are not consumed. A common issue is caused by a result set returned by SQL Server that includes each statement's row count. You can disable this by using SET NOCOUNT ON at the top of your query.

SQL Server Overall Performance

To analyze the overall performance of a SQL Server instance, besides examining hardware resource utilization, you should examine some general aspects of SQL Server. You can use the performance counters presented in Table 4-3.

Table 4-3. *Performance Monitor Counters to Analyze Generic SQL Pressure*

Object(Instance[,InstanceN])	Counter
SQLServer:Access Methods	FreeSpace Scans/sec Full Scans/sec Table Lock Escalations/sec Worktables Created/sec
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec Lock Wait Time (ms) Number of Deadlocks/sec
SQLServer:SQL Statistics	Batch Requests/sec SQL Re-Compilations/sec
SQLServer:General Statistics	Processes Blocked User ConnectionsTemp Tables Creation RateTemp Tables for Destruction

Let's break these down into different areas of concern to show the counters within the context where they would be more useful.

Missing Indexes

To analyze the possibility of missing indexes causing table scans or large data set retrievals, you can use the counter in Table 4-4.

Table 4-4. *Performance Monitor Counter to Analyze Excessive Data Scans*

Object(Instance[,InstanceN])	Counter
SQLServer:Access Methods	Full Scans/sec

Full Scans/Sec

This counter monitors the number of unrestricted full scans on base tables or indexes. Scans are not necessarily a bad thing. But they do represent a broader access of data, so they are likely to indicate a problem. A few of the main causes of a high Full Scans/sec value are as follows:

- Missing indexes
- Too many rows requested

- Not selective enough a predicate
- Improper T-SQL
- Data distribution or quantity doesn't support a seek

To further investigate queries producing these problems, use Extended Events to identify the queries (I will cover this tool in the next chapter). You can also retrieve this information from the Query Store (Chapter 11). Queries with missing indexes, too many rows requested, or badly formed T-SQL will have a large number of logical reads, caused by scanning the entire table or entire index and an increased CPU time.

Be aware that full scans may be performed for the temporary tables used in a stored procedure because most of the time you will not have indexes (or you will not need indexes) on temporary tables. Still, adding this counter to the baseline helps identify the possible increase in the use of temporary tables, which, when used inappropriately, can be bad for performance.

Dynamic Management Objects

Another way to check for missing indexes is to query the dynamic management view `sys.dm_db_missing_index_details`. This management view returns information that can suggest candidates for indexes based on the execution plans of the queries being run against the database. The view `sys.dm_db_missing_index_details` is part of a series of DMVs collectively referred to as the *missing indexes feature.* These DMVs are based on data generated from execution plans stored in the cache. You can query directly against this view to gather data to decide whether you want to build indexes based on the information available from within the view. Missing indexes will also be shown within the XML execution plan for a given query, but I'll cover that more in the next chapter. While these views are useful for suggesting possible indexes, since they can't be linked to a particular query, it can be unclear which of these indexes is most useful. You'll be better off using the techniques I show in the next chapter to associate a missing index with a particular query. For all the missing index suggestions, you must test them prior to implementing any suggestion on your systems.

The opposite problem to a missing index is one that is never used. The DMV `sys.dm_db_index_usage_stats` shows which indexes have been used, at least since the last restart of the SQL Server instance. Unfortunately, there are a number of ways that counters within this DMV get reset or removed, so you can't completely rely on it for a 100 percent accurate view of index use. You can also view the indexes in use with a lower-level DMV, `sys.dm_db_index_operational_stats`. It will help to show where indexes are slowing down because of contention or I/O. I'll cover these both in more detail in Chapter 20. You may also find that the suggestions from the Database Tuning Advisor (covered in Chapter 10) may be able to help you with specific indexes for specific queries.

Database Concurrency

To analyze the impact of database blocking on the performance of SQL Server, you can use the counters shown in Table 4-5.

Table 4-5. *Performance Monitor Counters to Analyze SQL Server Locking*

Object(Instance[,InstanceN])	Counter
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec
	Lock Wait Time (ms)
	Number of Deadlocks/sec

Total Latch Wait Time (Ms)

Latches are used internally by SQL Server to protect the integrity of internal structures, such as a table row, and are not directly controlled by users. This counter monitors total latch wait time (in milliseconds) for latch requests that had to wait in the last second. A high value for this counter can indicate that SQL Server is spending too much time waiting on its internal synchronization mechanism. For a detailed discussion, see the (older, but still relevant) white paper from Microsoft at `https://bit.ly/2wx4gAJ`.

Lock Timeouts/Sec and Lock Wait Time (Ms)

You should expect Lock Timeouts/sec to be 0 and Lock Wait Time (ms) to be very low. A nonzero value for Lock Timeouts/sec and a high value for Lock Wait Time (ms) indicate that excessive blocking is occurring in the database. Three approaches can be adopted in this case.

- You can identify the costly queries currently in cache using data from SQL Profiler or by querying `sys.dm_exec_query_stats`, and then you can optimize the queries appropriately.
- You can use blocking analysis to diagnose the cause of excessive blocking. It is usually advantageous to concentrate on optimizing the costly queries first because this, in turn, reduces blocking for others. In Chapter 20, you will learn how to analyze and resolve blocking.
- Extended Events supplies a blocking event called `blocked_process_report` that you can enable and set a threshold to capture blocking information. Extended Events will be covered in Chapter 6, and `blocked_process_report` will be addressed in Chapter 20.

Just remember that some degree of locks is a necessary part of the system. You'll want to establish a baseline to track thoroughly whether a given value is cause for concern.

Number of Deadlocks/Sec

You should expect to see a 0 value for this counter. If you find a nonzero value, then you should identify the victimized request and either resubmit the database request automatically or suggest that the user do so. More important, an attempt should be made to troubleshoot and resolve the deadlock. Chapter 21 shows how to do this.

Nonreusable Execution Plans

Since generating an execution plan for a stored procedure query requires CPU cycles, you can reduce the stress on the CPU by reusing the execution plan. To analyze the number of stored procedures that are recompiling, you can look at the counter in Table 4-6.

Table 4-6. *Performance Monitor Counter to Analyze Execution Plan Reusability*

Object(Instance[,InstanceN])	Counter
SQLServer:SOL Statistics	SOL Re-Compilations/sec

Recompilations of stored procedures add overhead on the processor. You want to see a value as close to 0 as possible for the SOL Re-Compilations/sec counter, but you won't ever see that. If you consistently see values that deviate from your baseline measures or that spike wildly, then you should use Extended Events to further investigate the stored procedures undergoing recompilations. Once you identify the relevant stored procedures, you should attempt to analyze and resolve the cause of recompilations. In Chapter 17, you will learn how to analyze and resolve various causes of recompilation.

General Behavior

SQL Server provides additional performance counters to track some general aspects of a SQL Server system. Table 4-7 lists a couple of the most commonly used counters.

Table 4-7. *Performance Monitor Counters to Analyze Volume of Incoming Requests*

Object(Instance[,InstanceN])	Counter
SQLServer:General Statistics	User Connections
SQLServer:SQL Statistics	Batch Requests/sec

User Connections

Multiple read-only SQL Server instances can work together in a load-balancing environment (where SQL Server is spread over several machines) to support a large number of database requests. In such cases, it is better to monitor the User Connections counter to evaluate the distribution of user connections across multiple SQL Server instances. User Connections can range all over the spectrum with normal application behavior. This is where a baseline is essential to determine the expected behavior. You will see how you can establish this baseline shortly.

Batch Requests/Sec

This counter is a good indicator of the load on SQL Server. Based on the level of system resource utilization and Batch Requests/sec, you can estimate the number of users SQL Server may be able to take without developing resource bottlenecks. This counter value, at different load cycles, helps you understand its relationship with the number of database connections. This also helps you understand SQL Server's relationship with Web Request/sec, that is, Active Server Pages.Requests/sec for web applications using Microsoft Internet Information Services (IIS) and Active Server Pages (ASP). All this analysis helps you better understand and predict system behavior as the user load changes.

The value of this counter can range over a wide spectrum with normal application behavior. A normal baseline is essential to determine the expected behavior.

Summary

In this chapter, you learned how to gather metrics on the CPU, the network, and SQL Server in general. All this information feeds into your ability to understand what's happening on your system before you delve into attempting to tune queries. Remember that the CPU is affected by the other resources since it's the thing that has to manage those resources, so some situations that can look like a CPU problem are better explained as a disk or memory issue. Networks are seldom a major bottleneck for SQL Server. You have a number of methods of observing SQL Server internals behavior through Performance Monitor counters, just like the other parts of the system. This concludes the discussion of the various system metrics. Next, you'll learn how to put all that together to create a baseline.

CHAPTER 5

Creating a Baseline

In the previous three chapters, you learned a lot about various possible system bottlenecks caused by memory, the disk, and the CPU. I also introduced a number of Performance Monitor metrics for gathering data on these parts of the system. Within the descriptions of most of the counters, I referred to comparing your metric to a baseline. This chapter will cover how to gather your metrics so that you have that baseline for later comparison. I'll go over how to configure an automated method of gathering this information. A baseline is a fundamental part of understanding system behavior, so you should always have one available. This chapter covers the following topics:

- Considerations for monitoring virtual and hosted machines
- How to set up an automated collection of Performance Monitor metrics
- Considerations to avoid issues when using Performance Monitor
- Baselines for Azure SQL Database
- Creating a baseline

Considerations for Monitoring Virtual and Hosted Machines

Before you start creating the baseline, I will talk about virtual machines (VMs). More and more SQL Server instances are running on VMs. When you are working with VMs or you are hosting VMs in remote environments such as Amazon or Microsoft Azure, many of the standard performance counters will no longer display accurate information. If you monitor these counters within the VM, your numbers may not be helpful from a troubleshooting perspective. If you monitor these counters on the physical box, assuming you have access to it, which doubtless is shared by multiple different VMs, you

G. Fritchey, *SQL Server 2017 Query Performance Tuning*, https://doi.org/10.1007/978-1-4842-3888-2_5

will be unable to identify specific SQL Server instance resource bottlenecks. Because of this, additional information must be monitored when working with a VM. Most of the information that you can gather on disk and network performance is still applicable within a VM setting. All query metric information will be accurate for those queries. How long a query runs and how many reads it has are exactly that, the length of time and volume of reads. Primarily you'll find the memory and CPU metrics that are completely different and quite unreliable.

This is because CPU and memory are shared between machines within a virtualized server environment. You may start a process on one CPU and finish it on another one entirely. Some virtual environments can actually change the memory allocated to a machine as that machine's demands for memory go up and down. With these kinds of changes, traditional monitoring just isn't applicable. The good news is that the major VM vendors provide you with guidance on how to monitor their systems and how to use SQL Server within their systems. You can largely rely on these third-party documents for the specifics of monitoring a VM. Taking the two most common hypervisors, VMware and HyperV, here is a document from each:

- VMware Monitoring Virtual Machine Performance (`http://bit.ly/1f37tEh`)
- Measuring Performance on HyperV (`http://bit.ly/2y2U6Iw`)

The queues counters, such as processor queue length, are still applicable when monitoring within a VM. These indicate that the VM itself is starved for resources, starving your SQL Server instance so that it has to wait for access to the virtual CPU. The important thing to remember is that CPU and memory are going to be potentially slower on a VM because the management of the VM is getting in the way of the system resources. You may also see slower I/O on a hosted VM because of the shared nature of hosted resources.

There's also a built-in, automated, baseline mechanism within Azure SQL Database and any instance of SQL Server 2016 or greater, known as the Query Store. We'll cover the Query Store in detail in Chapter 11.

Another mechanism available for understanding how the system is behaving are the DMVs. It's hard to consider them the same thing as a baseline since they change so much depending the cache, reboots, failovers, and other mechanisms. However, they do provide a way to see an aggregated view of query performance. We'll cover them more in Chapter 6 and throughout the rest of the book.

Creating a Baseline

Now that you have looked at a few of the main performance counters, let's see how to bring these counters together to create a system baseline. These are the steps you need to follow:

1. Create a reusable list of performance counters.
2. Create a counter log using your list of performance counters.
3. Minimize Performance Monitor overhead.

Creating a Reusable List of Performance Counters

Run the Performance Monitor tool on a Windows Server 2016 machine connected to the same network as that of the SQL Server system. Add performance counters to the View Chart display of the Performance Monitor through the Properties ➤ Data ➤ Add Counters dialog box, as shown in Figure 5-1.

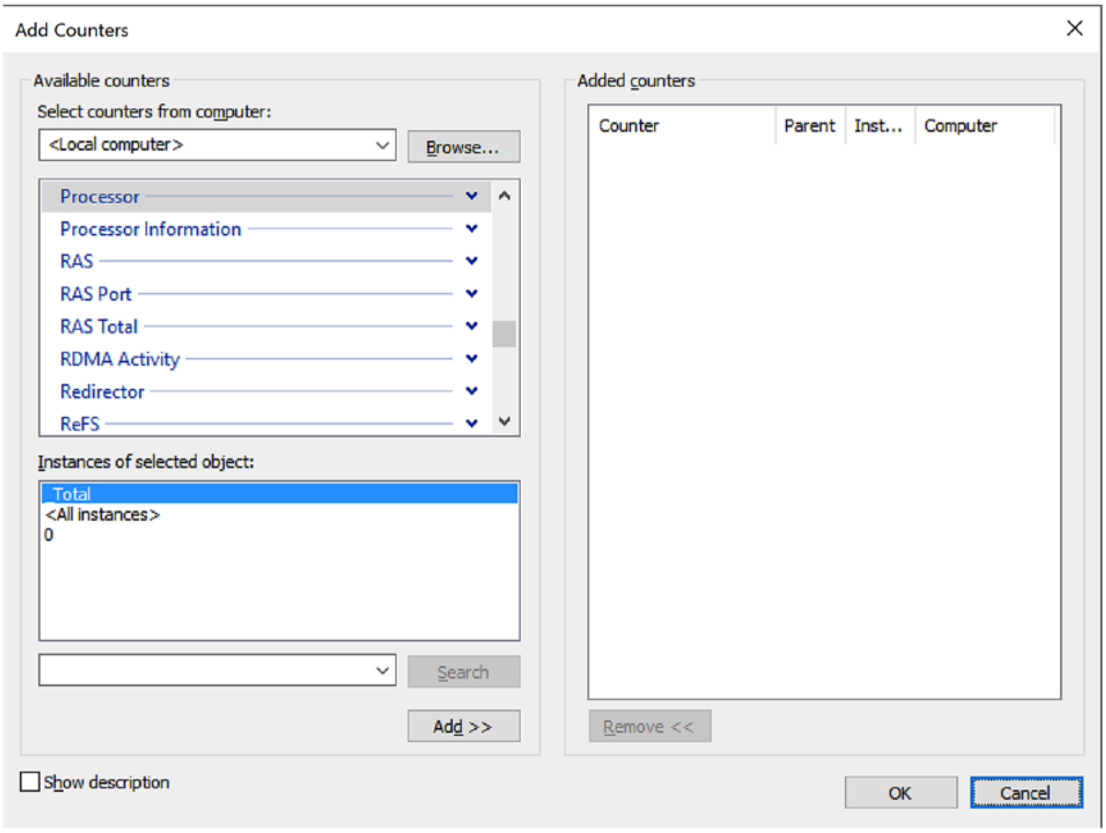


Figure 5-1. *Adding Performance Monitor counters*

For example, to add the performance counter SQLServer:Latches:Total Latch Wait Time(ms), follow these steps:

1. Select the option Select Counters from Computer and specify the computer name running SQL Server in the corresponding entry field, or, when running Performance Monitor locally, you'll see "<Local Computer>" like in Figure 5-1.
2. Click the arrow next to the performance object SQLServer:Latches.
3. Choose the Total Latch Wait Time(ms) counter from the list of performance counters.
4. Click the Add button to add this performance counter to the list of counters to be added.
5. Continue as needed with other counters. When finished, click the OK button.

When creating a reusable list for your baseline, you can repeat the preceding steps to add all the performance counters listed in Table 5-1.

Table 5-1. *Performance Monitor Counters to Analyze SQL Server Performance*

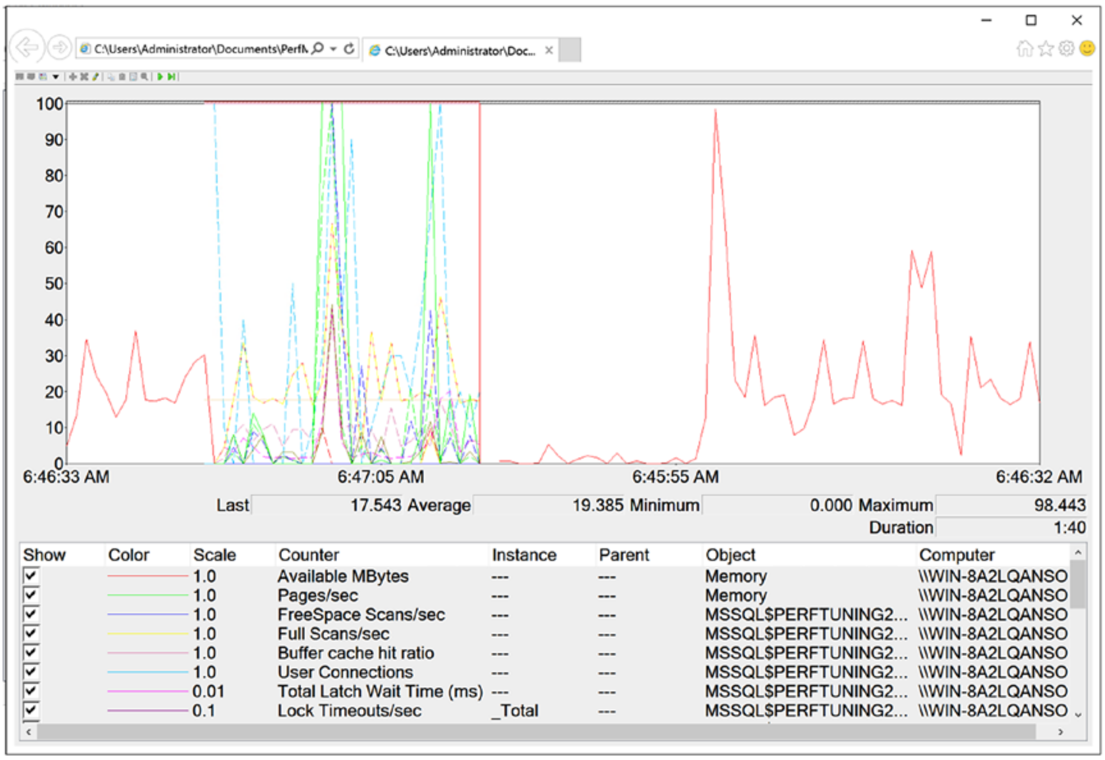
Object(Instance[,InstanceN])	Counter
Memory	Available MBytes Pages/sec
PhysicalDisk(Data-disk, Log-disk)	% Disk TimeCurrent Disk Queue Length Disk Transfers/sec Disk Bytes/sec
Processor(_Total)	% Processor Time % Privileged Time
System	Processor Queue Length Context Switches/sec
Network Interface(Network card)	Bytes Total/sec
Network Segment	% Net Utilization
SQLServer:Access Methods	FreeSpace Scans/sec Full Scans/sec
SQLServer:Buffer Manager	Buffer cache hit ratio
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec Lock Wait Time (ms) Number of Deadlocks/sec
SQLServer:Memory Manager	Memory Grants Pending Target Server Memory (KB) Total Server Memory (KB)
SQLServer:SQL Statistics	Batch Requests/sec SQL Re-Compilations/sec
SQLServer:General Statistics	User Connections

Once you have added all the performance counters, close the Add Counters dialog box by clicking OK. To save the list of counters as an `.htm` file, right-click anywhere in the right frame of Performance Monitor and select the Save Settings As menu item.

The `.htm` file lists all the performance counters that can be used as a base set of counters to create a counter log or to view Performance Monitor graphs interactively for the same SQL Server machine. To use this list of counters for other SQL Server machines, open the `.htm` file in an editor such as Notepad and replace all instances of `\\SQLServerMachineName` with nothing (just a blank string) .

A shortcut to all this is outlined by Erin Stellato in the article "Customizing the Default Counters for Performance Monitor" (`http://bit.ly/1brQKeZ`). There's also an easier way to deal with some of this data using a tool supplied by Microsoft, Performance Analysis of Logs (PAL), available at `https://bit.ly/2KeJJmy`.

You can also use this counter list file to view Performance Monitor graphs interactively in an Internet browser, as shown in Figure 5-2.


C:\Users\Administrator\Documents\PerfM
C:\Users\Administrator\Doc...
100
90
80
70
60
50
40
30
20
10
0
6:46:33 AM
6:47:05 AM
6:45:55 AM
6:46:32 AM

Last 17.543 Average 19.385 Minimum 0.000 Maximum 98.443
Duration 1:40

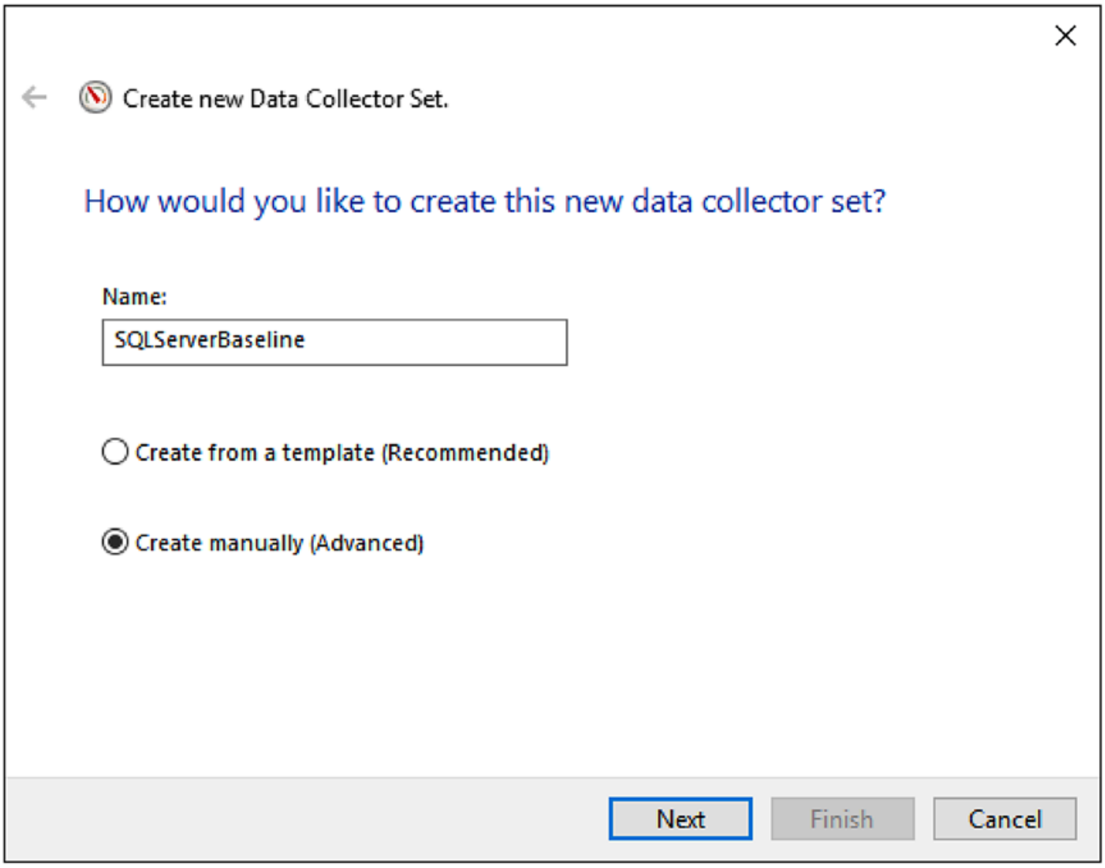
Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	🟥	1.0	Available MBytes	---	---	Memory	\\WIN-8A2LQANSO
✓	🟩	1.0	Pages/sec	---	---	Memory	\\WIN-8A2LQANSO
✓	🟦	1.0	FreeSpace Scans/sec	---	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO
✓	🟨	1.0	Full Scans/sec	---	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO
✓	🟥	1.0	Buffer cache hit ratio	---	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO
✓	🟦	1.0	User Connections	---	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO
✓	🟪	0.01	Total Latch Wait Time (ms)	---	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO
✓	🟪	0.1	Lock Timeouts/sec	_Total	---	MSSQL$PERFTUNING2...	\\WIN-8A2LQANSO

Figure 5-2. *Performance Monitor in Internet browser*

Creating a Counter Log Using the List of Performance Counters

Performance Monitor provides a counter log facility to save the performance data of multiple counters over a period of time. You can view the saved counter log using Performance Monitor to analyze the performance data. It is usually convenient to create a counter log from a defined list of performance counters. Simply collecting the data rather than viewing it through the GUI is the preferred method of automation to prepare for troubleshooting your server's performance or establishing a baseline.

Within Performance Monitor, expand Data Collector Sets ➤ User Defined. Right-click and select New ➤ Data Collector Set. Define the name of the set and make this a manual creation by clicking the appropriate radio button; then click Next just like I configured Figure 5-3.


Create new Data Collector Set.

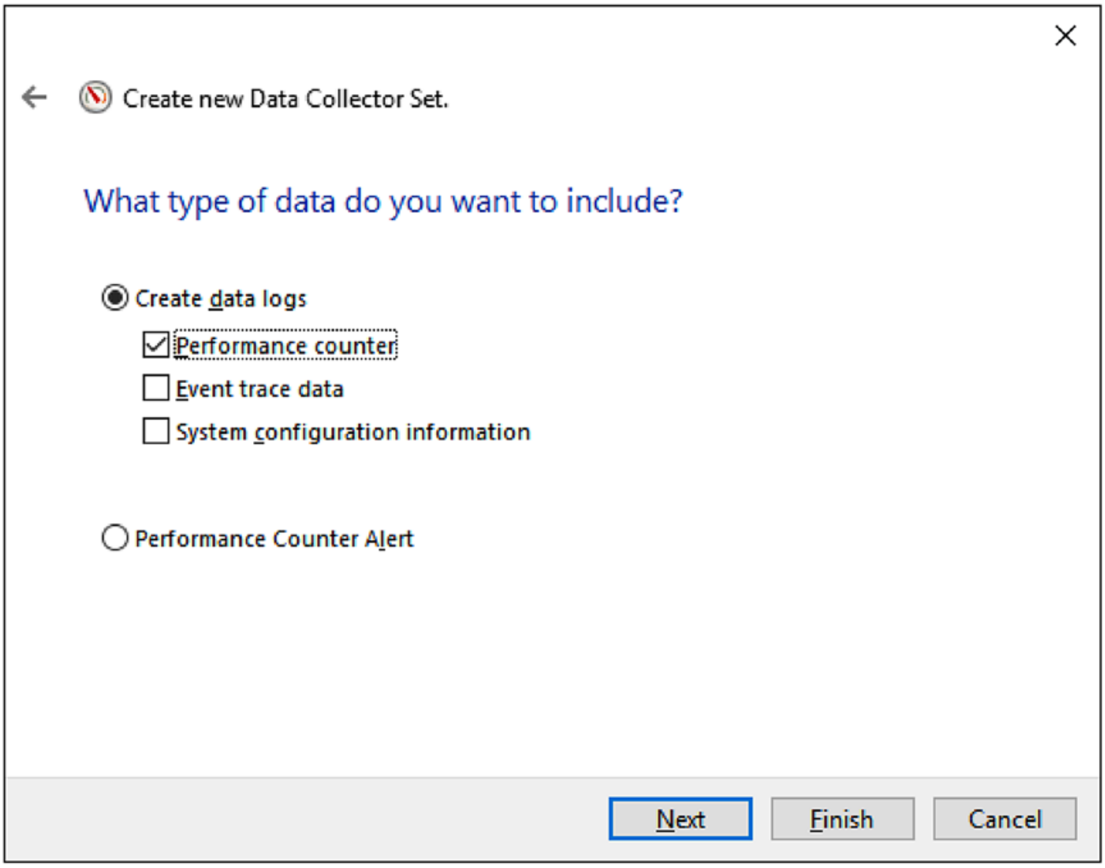
How would you like to create this new data collector set?

Name: SQLServerBaseline

Create from a template (Recommended) [✓ Create manually (Advanced)]

Figure 5-3. *Naming the data collector set*

You'll have to define what type of data you're collecting. In this case, select the check box Performance Counters under the Create Data Logs radio button and then click Next, as shown in Figure 5-4.



Create new Data Collector Set.

What type of data do you want to include?

(●) Create data logs

[✓] Performance counter

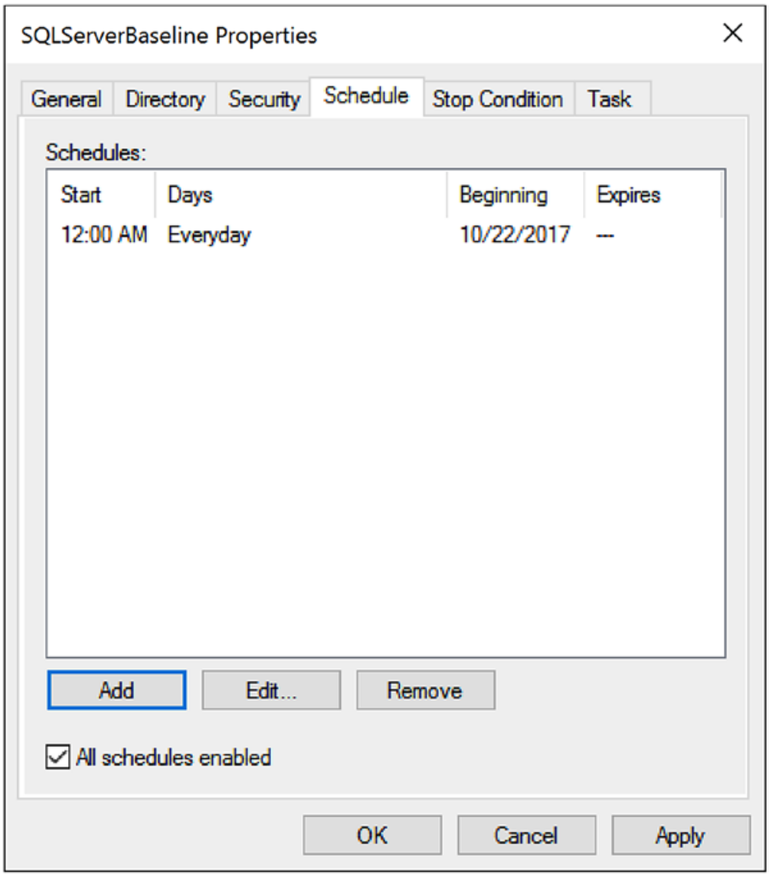
[ ] Event trace data

[ ] System configuration information

( ) Performance Counter Alert

Figure 5-4. *Selecting data logs and performance counters for the data collector set*

Here you can define the performance objects you want to collect using the same Add Counters dialog box shown earlier in Figure 5-1. Clicking Next allows you to define the destination folder. Click Next, then select the radio button Open Properties for This Data Collector Set, and click Finish. You can schedule the counter log to automatically start at a specific time and stop after a certain time period or at a specific time. You can configure these settings through the Schedule pane. You can see an example in Figure 5-5.



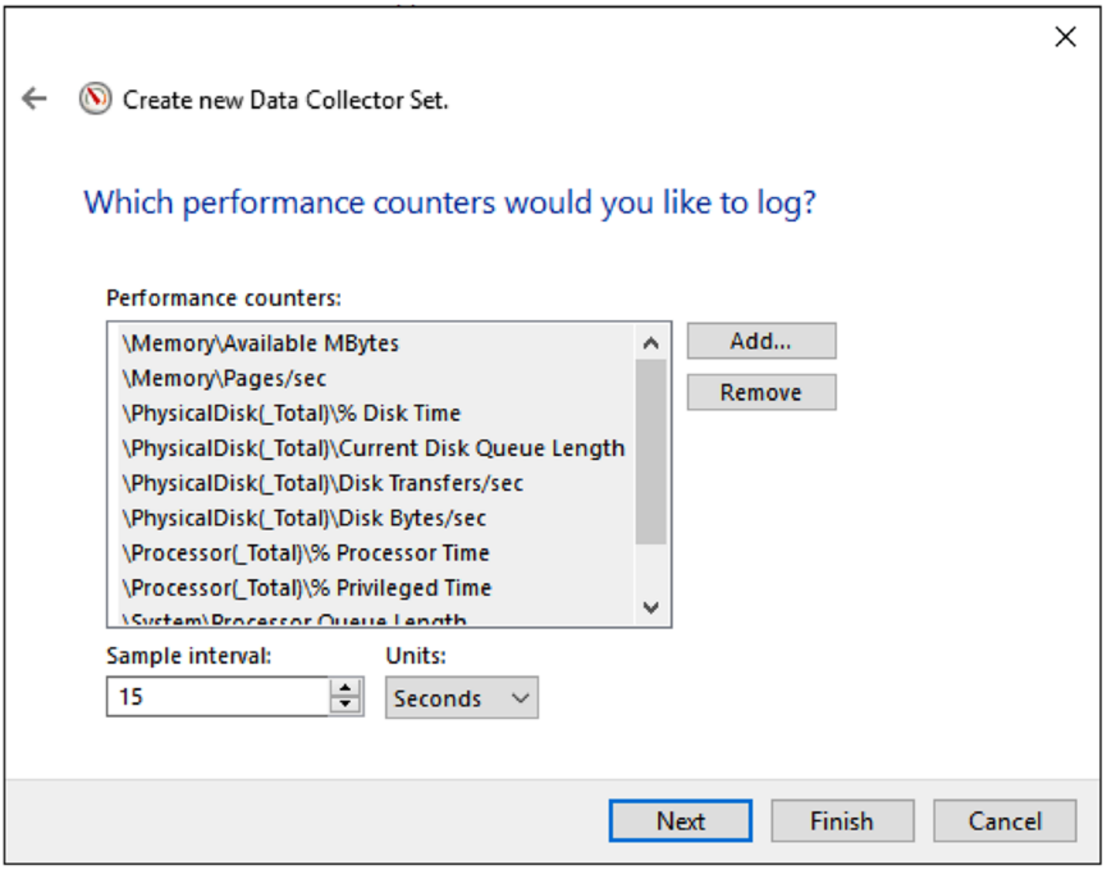
Schedules:

Start	Days	Beginning	Expires
12:00 AM	Everyday	10/22/2017	---

☑ All schedules enabled

Figure 5-5. *A schedule defined in the properties of the data collector set*

Figure 5-6 summarizes which counters have been selected as well as the frequency with which the counters will be collected.


Create new Data Collector Set.

Which performance counters would you like to log?

Performance counters:

\Memory\Available MBytes
\Memory\Pages/sec
\PhysicalDisk(_Total)\% Disk Time
\PhysicalDisk(_Total)\Current Disk Queue Length
\PhysicalDisk(_Total)\Disk Transfers/sec
\PhysicalDisk(_Total)\Disk Bytes/sec
\Processor(_Total)\% Processor Time
\Processor(_Total)\% Privileged Time
\System\Processor Queue Length

Sample interval: 15

Units: Seconds

Figure 5-6. *Defining a Performance Monitor counter log*

Note I'll offer additional suggestions for these settings in the section that follows.

For additional information on how to create counter logs using Performance Monitor, please refer to the Microsoft Knowledge Base article "Performance Tuning Guidelines for Windows Server 2016" (`http://bit.ly/1icVvgn`).

Performance Monitor Considerations

The Performance Monitor tool is designed to add as little overhead as possible, if used correctly. To minimize the impact of using this tool on a system, consider the following suggestions:

- Limit the number of counters, specifically performance objects.
- Use counter logs instead of viewing Performance Monitor graphs interactively.
- Run Performance Monitor remotely while viewing graphs interactively.
- Save the counter log file to a different local disk.
- Increase the sampling interval.

Let's consider each of these points in more detail.

Limit the Number of Counters

Monitoring large numbers of performance counters with small sampling intervals could incur some amount of overhead on the system. The bulk of this overhead comes from the number of performance objects you are monitoring, so selecting them wisely is important. The number of counters for the selected performance objects does not add much overhead because it gives only an attribute of the object itself. Therefore, it is important to know what objects you want to monitor and why.

Prefer Counter Logs

Use counter logs instead of viewing a Performance Monitor graph interactively because Performance Monitor graphing is more costly in terms of overhead. Monitoring current activities should be limited to short-term viewing of data, troubleshooting, and diagnosis. Performance data reported via a counter log is *sampled*, meaning that data is collected periodically rather than traced, whereas the Performance Monitor graph is updated in real time as events occur. Using counter logs will reduce that overhead.

View Performance Monitor Graphs Remotely

Since viewing the live performance data using Performance Monitor graphs creates a fair amount of overhead on the system, run the tool remotely on a different machine and connect to the SQL Server system through the tool. To remotely connect to the SQL Server machine, run the Performance Monitor tool on a machine connected to the network to which the SQL Server machine is also connected.

Type the computer name (or IP address) of the SQL Server machine in the Select Counters from Computer box. Be aware that if you connect to the production server through a Windows Server 2016 terminal service session, the major part of the tool will still run on the server.

However, I still encourage you to avoid using the Performance Monitor graphs for viewing live data. You can use the graphs to look at the files collected through counter logs and should have a bias toward using those logs.

Save Counter Log Locally

Collecting the performance data for the counter log does not incur the overhead of displaying any graph. So, while using counter log mode, it is more efficient to log counter values locally on the SQL Server system instead of transferring the performance data across the network. Put the counter log file on a local disk other than the ones that are monitored, meaning your SQL Server data and log files.

Then, after you collect the data, copy that counter log to your local machine to analyze it. That way, you're working only on a copy, and you're not adding I/O overhead to your storage location.

Increase the Sampling Interval

Because you are mainly interested in the resource utilization pattern during baseline monitoring, you can easily increase the performance data sampling interval to 60 seconds or more to decrease the log file size and reduce demand on disk I/Os. You can use a short sampling interval to detect and diagnose timing issues. Even while viewing Performance Monitor graphs interactively, increase the sampling interval from the default value of one second per sample. Just remember, changing the sampling size up or down can affect the granularity of the data as well as the quantity. You have to weigh these choices carefully.

System Behavior Analysis Against Baseline

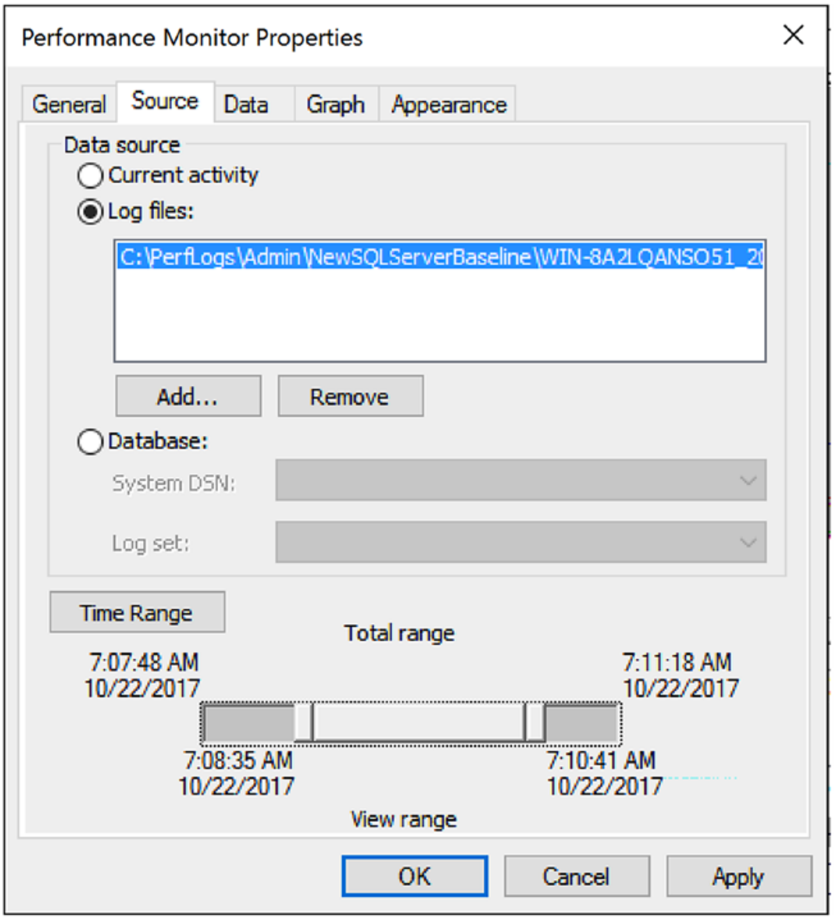
The default behavior of a database application changes over time because of various factors such as the following:

- Data volume and distribution changes
- Increased user base
- Change in usage pattern of the application
- Additions to or changes in the application's behavior
- Installation of new service packs or software upgrades
- Changes to hardware

Because of these changes, the baseline created for the database server slowly loses its significance. It may not always be accurate to compare the current behavior of the system with an old baseline. Therefore, it is important to keep the baseline current by creating a new baseline at regular time intervals. It is also beneficial to archive the previous baseline logs so that they can be referred to later, if required. So while, yes, older baselines are not applicable to day-to-day operations, they do help you in establishing patterns and long-term trends.

The counter log for the baseline or the current behavior of the system can be analyzed using the Performance Monitor tool by following these steps:

1. Open the counter log. Use Performance Monitor's toolbar item View Log File Data and select the log file's name.
2. Add all the performance counters to analyze the performance data. Note that only the performance objects, counters, and instances selected during the counter log creation are shown in the selection lists.
3. Analyze the system behavior at different parts of the day by adjusting the time range accordingly, as shown in Figure 5-7.


Performance Monitor Properties

General | Source | Data | Graph | Appearance

Data source
- ( ) Current activity
- (●) Log files:

```
C:\PerfLogs\Admin\NewSQLServerBaseline\WIN-8A2LQANSO51_20
```

Add... | Remove

( ) Database:
System DSN:
Log set:

Time Range

Total range: 7:07:48 AM 10/22/2017 – 7:11:18 AM 10/22/2017

View range: 7:08:35 AM 10/22/2017 – 7:10:41 AM 10/22/2017

OK | Cancel | Apply

Figure 5-7. *Defining time range for log analysis*

During a performance review, you can analyze the system-level behavior of the database by comparing the current value of performance counters with the latest baseline. Take the following considerations into account while comparing the performance data:

- Use the same set of performance counters in both cases.
- Compare the minimum, maximum, and average values of the counters as applicable for the individual counters. I explained the specific values for the counters earlier.

- Some counters have an absolute good/bad value, as mentioned previously. The current value of these counters need not be compared with the baseline values. For example, if the current average value of the Deadlocks/min counter is 10, it indicates that the system is suffering from a large number of deadlocks. Even though it does not require a comparison with the baseline, it is still advantageous to review the corresponding baseline value because your deadlock issues might have existed for a long time. Having the archived baseline logs helps detect the evolving occurrence of the deadlock.
- Some counters do not have a definitive good/bad value. Because their value depends on the application, a relative comparison with the corresponding baseline counters is a must. For example, the current value of the User Connections counter for SQL Server does not signify anything good or bad with the application. But comparing it with the corresponding baseline value may reveal a big increase in the number of user connections, indicating an increase in the workload.
- Compare a range of values for the counters from the current and the baseline counter logs. The fluctuation in the individual values of the counters will be normalized by the range of values.
- Compare logs from the same part of the day. For most applications, the usage pattern varies during different parts of the day. To obtain the minimum, maximum, and average values of the counters for a specific time, adjust the time range of the counter logs, as shown previously.

Once the system-level bottleneck is identified, the internal behavior of the application should be analyzed to determine the cause of the bottleneck. Identifying and optimizing the source of the bottleneck will help use the system resources efficiently.

Baseline for Azure SQL Database

Just as you want to have a baseline for your SQL Server instances running on physical boxes and VMs, you need to have a baseline for the performance of Azure SQL Databases. You can't capture Performance Monitor metrics for this. Also, Azure SQL Database is not represented as a virtual machine or physical server. It's a database as a service. As such, you don't measure CPU or disk usage. Instead, Microsoft has defined a unit of performance measure known as the Database Transaction Unit (DTU). You can observe the DTU behavior of your database over time.

The DTU is defined as a blended measure of I/O, CPU, and memory. It does not represent literal transactions as the name might imply but is instead a measure of the performance of a database within the service. You can query `sys.resource_stats` as a way to see CPU usage and the storage data. It retains a 14-day running history and aggregates the data over five-minute intervals.

While the Azure Portal provides a mechanism for observing the DTU use, it doesn't provide you with a mechanism for establishing a baseline. Instead, you should use the Azure SQL Database–specific DMV `sys.dm_db_resource_stats`. This DMV maintains information about the DTU usage of a given Azure SQL Database. It contains one hour of information in 15-minute aggregates. To establish a baseline as with a SQL Server instance, you would need to capture this data over time. Collecting the information displayed within `sys.dm_db_resource_stats` into a table would be how you could establish a baseline for the performance metrics of your Azure SQL Database.

Azure SQL Database has the Query Store enabled by default, so you can use that to understand what's happening on the system.

Summary

In this chapter, you learned how to use the Performance Monitor tool to analyze the overall behavior of SQL Server as well as the effect of a slow-performing database application on system resources. You also learned about the establishment of baselines as part of your monitoring of the servers and databases. With these tools you'll be able to understand when you're experiencing deviations from that standard behavior. You'll want to collect a baseline on a regular basis so that the data doesn't get stale.

In the next chapter, you will learn how to analyze the workload of a database application for performance tuning.

CHAPTER 6

Query Performance Metrics

A common cause of slow SQL Server performance is a heavy database application workload—the nature and quantity of the queries themselves. Thus, to analyze the cause of a system bottleneck, it is important to examine the database application workload and identify the SQL queries causing the most stress on system resources. To do this, you can use Extended Events and other Management Studio tools.

In this chapter, I cover the following topics:

- The basics of Extended Events
- How to analyze SQL Server workload and identify costly SQL queries using Extended Events
- How to track query performance through dynamic management objects

Extended Events

Extended Events was introduced in SQL Server 2008, but with no GUI in place and a reasonably complex set of code to set it up, Extended Events wasn't used much to capture performance metrics. With SQL Server 2012, a GUI for managing Extended Events was introduced, taking away the final issue preventing Extended Events from becoming the preferred mechanism for gathering query performance metrics as well as other metrics and measures. Trace events, previously the best mechanism for gathering these metrics, are in deprecation and are not actively under development. No new trace events have been added for years. Profiler, the GUI for generating and consuming trace events, can even create performance problems if you run it inappropriately against a production instance. As a result, the examples in the book will be using Extended Events primarily and the Query Store as a secondary mechanism (Query Store is covered in Chapter 11).

G. Fritchey, *SQL Server 2017 Query Performance Tuning*, https://doi.org/10.1007/978-1-4842-3888-2_6

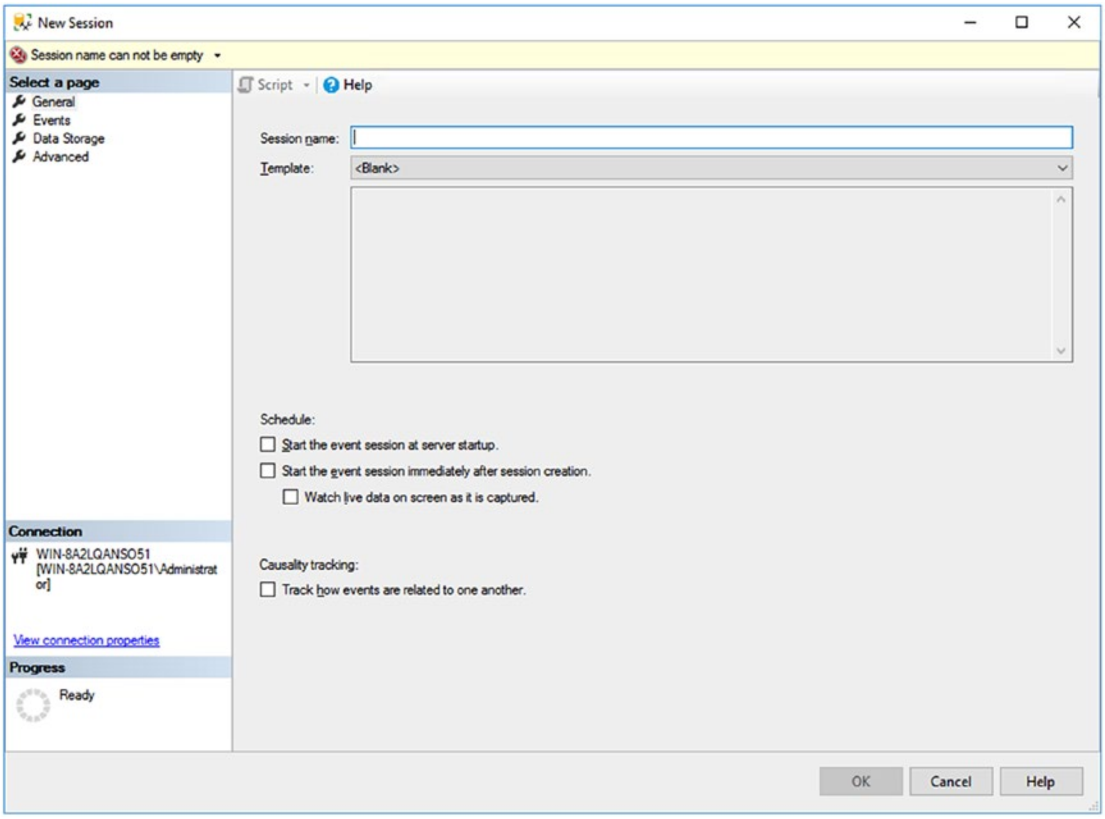
Extended Events allows you to do the following:

- Graphically monitor SQL Server queries
- Collect query information in the background
- Analyze performance
- Diagnose problems such as deadlocks
- Debug a Transact-SQL (T-SQL) statement

You can also use Extended Events to capture other sorts of activities performed on a SQL Server instance. You can set up Extended Events from the graphical front end or through direct T-SQL calls to the procedures. The most efficient way to define an Extended Events session is through the T-SQL commands, but a good place to start learning about sessions is through the GUI.

Extended Events Sessions

You will find the Extended Events tooling in the Management Studio GUI. You can navigate using the Object Explorer to the Management folder on a given instance to find the Extended Events folder. From there you can look at sessions that have already been built on the system. To start setting up your own sessions, just right-click the Sessions folder and select New Session. There is a wizard available for setting up sessions, but it doesn't do anything the regular GUI doesn't do, and the regular GUI is easy to use. A window opens to the first page, called General, as shown in Figure 6-1.



New Session

Session name can not be empty

Select a page
- General
- Events
- Data Storage
- Advanced

Script · Help

Session name:

Template: <Blank>

Schedule:
- ☐ Start the event session at server startup.
- ☐ Start the event session immediately after session creation.
  - ☐ Watch live data on screen as it is captured.

Causality tracking:
- ☐ Track how events are related to one another.

Connection

WIN-8A2LQANSO51 [WIN-8A2LQANSO51\Administrator]

View connection properties

Progress

Ready

OK · Cancel · Help

Figure 6-1. *Extended Events New Session window, General page*

You will have to supply a session name. I strongly suggest giving it a clear name so you know what the session is doing when you check it later. You also have the choice of using a template. Templates are predefined sessions that you can put to work with minimal effort. There are five templates immediately associated with query tuning, under the Query Execution category:

- *Query Batch Sampling*: This template will capture queries and procedure calls for 20 percent of all active sessions on the server.
- *Query Batch Tracking*: This template captures all queries and procedures for all sessions on the server.
- *Query Detail Sampling*: This template contains a set of events that will capture every statement in queries and procedures for 20 percent of all active sessions on the server.

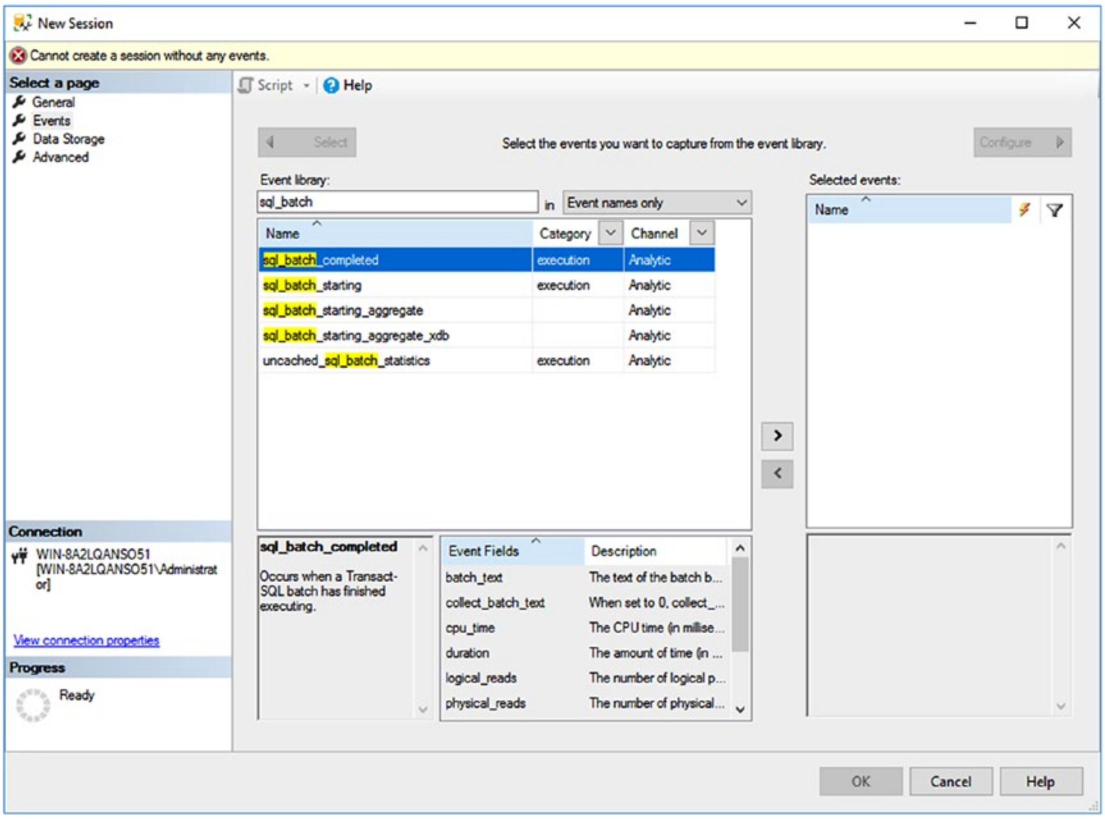
- *Query Detail Tracking*: This template is the same as Query Batch Tracking, but for every single statement in the system as well. This generates a large amount of data.
- *Query Wait Statistic*: This template captures wait statistics for each statement of every query and procedure for 20 percent of all active sessions.

Further, there are templates that emulate the ones you're used to having from Profiler. Also, introduced in SQL Server 2017, there is one additional method for quickly looking at query performance with minimal effort. At the bottom of the Object Explorer pane is a new folder, XE Profiler. Expanding the folder you'll find two Extended Events sessions that define query monitoring similar to what you would normally see within Profiler. I'll cover the Live Data window, which these options open, later in the chapter. Instead of launching into this, you'll skip the templates and the XE Profiler reports to set up your own events so you can see how it's done.

Note Nothing is free or without risk. Extended Events is a much more efficient mechanism for gathering information about the system than the old trace events. Extended Events is not without cost and risk. Depending on the events you define and, even more, on some of the global fields that I discuss in more detail later in the chapter, you may see an impact on your system by implementing Extended Events. Exercise caution when using these events on your production system to ensure you don't cause a negative impact. The Query Store can provide a lot of information for less impact, and you get even less impact using the DMOs (detailed later in this chapter). Those alternatives can work in some situations.

Looking at the first page of the New Session window, in addition to naming the session, there are a number of other options. You must decide whether you want the session to start when the server starts. Collecting performance metrics over a long period of time generates lots of data that you'll have to deal with. You can also decide whether you'd like to start this session immediately after you create it and whether you want to watch live data. Finally, the last option is to determine whether you want to track event causality. We'll address this later in the chapter.

As you can see, the New Session window is actually pretty close to already being a wizard. It just lacks a Next button. Once you've provided a name and made the other choices here, click the next page on the left of the window, Events, as shown in Figure 6-2.



Cannot create a session without any events.

Select the events you want to capture from the event library.

Event library: sql_batch in Event names only

Name	Category	Channel
sql_batch_completed	execution	Analytic
sql_batch_starting	execution	Analytic
sql_batch_starting_aggregate		Analytic
sql_batch_starting_aggregate_xdb		Analytic
uncached_sql_batch_statistics	execution	Analytic

Selected events:

Name

sql_batch_completed

Occurs when a Transact-SQL batch has finished executing.

Event Fields	Description
batch_text	The text of the batch b...
collect_batch_text	When set to 0, collect_...
cpu_time	The CPU time (in millise...
duration	The amount of time (in ...
logical_reads	The number of logical p...
physical_reads	The number of physical...

Connection: WIN-8A2LQANSO51 [WIN-8A2LQANSO51\Administrator]

***Figure 6-2.** Extended Events New Session window, Events page*

An *event* represents various activities performed in SQL Server and, in some cases, the underlying operating system. There's an entire architecture around event targets, event packages, and event sessions, but the use of the GUI means you don't have to worry about all those details. I will cover some of the architecture when showing how to script a session later in this chapter.

For performance analysis, you are mainly interested in the events that help you judge levels of resource stress for various activities performed on SQL Server. By *resource stress,* I mean things such as the following:

- What kind of CPU utilization was involved for the T-SQL activity?
- How much memory was used?
- How much I/O was involved?
- How long did the SQL activity take to execute?
- How frequently was a particular query executed?
- What kind of errors and warnings were faced by the queries?

You can calculate the resource stress of a SQL activity after the completion of an event, so the main events you use for performance analysis are those that represent the completion of a SQL activity. Table 6-1 describes these events.

Table 6-1. *Events to Monitor Query Completion*

Event Category	Event	Description
Execution	`rpc_completed`	A remote procedure call completion event
	`sp_statement_completed`	A SQL statement completion event within a stored procedure
	`sql_batch_completed`	A T-SQL batch completion event
	`sql_statement_completed`	A T-SQL statement completion event

An RPC event indicates that the stored procedure was executed using the Remote Procedure Call (RPC) mechanism through an OLEDB command. If a database application executes a stored procedure using the T-SQL `EXECUTE` statement, then that stored procedure is resolved as a SQL batch rather than as an RPC.

A *T-SQL batch* is a set of SQL queries that are submitted together to SQL Server. A T-SQL batch is usually terminated by a `GO` command. The `GO` command is not a T-SQL statement. Instead, the `GO` command is recognized by the `sqlcmd` utility, as well as by Management Studio, and it signals the end of a batch. Each SQL query in the batch is considered a T-SQL statement. Thus, a T-SQL batch consists of one or more

T-SQL statements. Statements or T-SQL statements are also the individual, discrete commands within a stored procedure. Capturing individual statements with the `sp_statement_completed` or `sql_statement_completed` event can be a more expensive operation, depending on the number of individual statements within your queries. Assume for a moment that each stored procedure within your system contains one, and only one, T-SQL statement. In this case, the cost of collecting completed statements is very low, both for impact on the behavior of the system while collecting the data and on the amount of storage you need to collect the data. Now assume you have multiple statements within your procedures and that some of those procedures are calls to other procedures with other statements. Collecting all this extra data now becomes a more noticeable load on the system. The impact of capturing statements completely depends on the size and number of statements you are capturing. Statement completion events should be collected judiciously, especially on a production system. You should apply filters to limit the returns from these events. Filters are covered later in this chapter.

To add an event to the session, find the event in the Event library. This is simple; you can just type the name. In Figure 6-2 you can see `sql_batch` typed into the search box and that part of the event name highlighted. Once you have an event, use the arrow buttons to move the event from the library to the Selected Events list. To remove events not required, click the arrow to move it back out of the list and into the library.

Although the events listed in Table 6-1 represent the most common events used for determining query performance, you can sometimes use a number of additional events to diagnose the same thing. For example, as mentioned in Chapter 1, repeated recompilation of a stored procedure adds processing overhead, which hurts the performance of the database request. The execution category in the Event library includes an event, `sql_statement_recompile`, to indicate the recompilation of a statement (this event is explained in depth in Chapter 12). The Event library contains additional events to indicate other performance-related issues with a database workload. Table 6-2 shows a few of these events.

Table 6-2. *Events for Query Performance*

Event Category	Event	Description
Session	`login` `logout`	Keeps track of database connections when users connect to and disconnect from SQL Server.
	`existing_connection`	Represents all the users connected to SQL Server before the session was started.
Errors	`attention`	Represents the intermediate termination of a request caused by actions such as query cancellation by a client or a broken database connection including timeouts.
	`error_reported`	Occurs when an error is reported.
	`execution_warning`	Indicates a wait for a memory grant for a statement has lasted longer than a second or a memory grant for a statement has failed.
	`hash_warning`	Indicates the occurrence of insufficient memory in a hashing operation. Combine this with capturing execution plans to understand which operation had the error.
Warnings	`missing_column_statistics`	Indicates that the statistics of a column, which are statistics required by the optimizer to decide a processing strategy, are missing.
	`missing_join_predicate`	Indicates that a query is executed with no joining predicate between two tables.
	`sort_warnings`	Indicates that a sort operation performed in a query such as `SELECT` did not fit into memory.
Lock	`lock_deadlock`	Occurs when a process is chosen as a deadlock victim.
	`lock_deadlock_chain`	Shows a trace of the chain of queries creating the deadlock.
	`lock_timeout`	Signifies that the lock has exceeded the timeout parameter, which is set by SET LOCK_TIMEOUT timeout_period(ms).

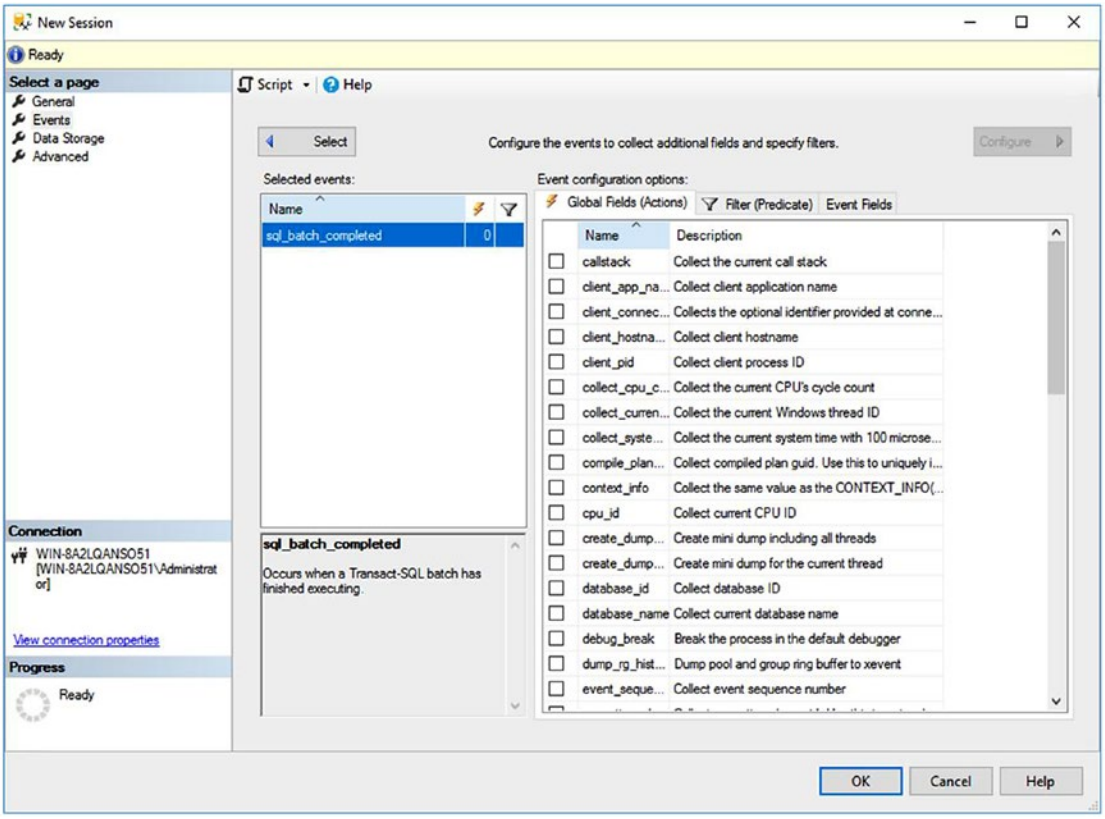
(*continued*)

Table 6-2. *(continued)*

Event Category	Event	Description
Execution	`sql_statement_recompile`	Indicates that an execution plan for a query statement had to be recompiled because one did not exist, a recompilation was forced, or the existing execution plan could not be reused. This is at the statement level, not the batch level, regardless of whether the batch is an ad hoc query stored procedure or prepared statements.
	`rpc_starting`	Represents the starting of a stored procedure. This is useful to identify procedures that started but could not finish because of an operation that caused an Attention event.
	`Query_post_compilation_showplan`	Shows the execution plan after a SQL statement has been compiled.
	`Query_post_execution_showplan`	Shows the execution plan after the SQL statement has been executed that includes execution statistics. Note, this event can be quite costly, so use it extremely sparingly and for short periods of time with good filters in place.
Transactions	`sql_transaction`	Provides information about a database transaction, including information such as when a transaction starts, completes, and rolls back.

Global Fields

Once you've selected the events that are of interest on the Events page, you may need to configure some settings, such as global fields. On the Events screen, click the Configure button. This will change the view of the Events screen, as shown in Figure 6-3.



Configure the events to collect additional fields and specify filters.

Selected events:

Name		
sql_batch_completed	0	

sql_batch_completed

Occurs when a Transact-SQL batch has finished executing.

Connection: WIN-8A2LQANSO51 [WIN-8A2LQANSO51\Administrator]

View connection properties

Progress: Ready

Event configuration options: Global Fields (Actions) | Filter (Predicate) | Event Fields

	Name	Description
☐	callstack	Collect the current call stack
☐	client_app_na...	Collect client application name
☐	client_connec...	Collects the optional identifier provided at conne...
☐	client_hostna...	Collect client hostname
☐	client_pid	Collect client process ID
☐	collect_cpu_c...	Collect the current CPU's cycle count
☐	collect_curren...	Collect the current Windows thread ID
☐	collect_syste...	Collect the current system time with 100 microse...
☐	compile_plan...	Collect compiled plan guid. Use this to uniquely i...
☐	context_info	Collect the same value as the CONTEXT_INFO(...
☐	cpu_id	Collect current CPU ID
☐	create_dump...	Create mini dump including all threads
☐	create_dump...	Create mini dump for the current thread
☐	database_id	Collect database ID
☐	database_name	Collect current database name
☐	debug_break	Break the process in the default debugger
☐	dump_rg_hist...	Dump pool and group ring buffer to xevent
☐	event_seque...	Collect event sequence number

Figure 6-3. *Global Fields selection in the Configure part of the Events page*

The global fields, called *actions* in T-SQL, represent different attributes of an event, such as the user involved with the event, the execution plan for the event, some additional resource costs of the event, and the source of the event. These are additional pieces of information that can be collected with an event. They add overhead to the collection of the event. Each event has a set of data it collects, which I'll talk about later in the chapter, but this is your chance to add more. Most of the time, when I can, I avoid this overhead for most data collection. But sometimes, there is information here you'll want to collect.

To add an action, just click the check box in the list provided on the Global Fields page shown in Figure 6-3. You can use additional data columns from time to time to diagnose the cause of poor performance. For example, in the case of a stored procedure recompilation, the event indicates the cause of the recompile through the `recompile_cause` event field. (This field is explained in depth in Chapter 18.) A few of the commonly used additional actions are as follows:

- `plan_handle`
- `query_hash`
- `query_plan_hash`
- `database_id`
- `client_app_name`
- `transaction_id`
- `session_id`

Other information is available as part of the event fields. For example, the `binary_data` and `integer_data` event fields provide specific information about a given SQL Server activity. For instance, in the case of a cursor, they specify the type of cursor requested and the type of cursor created. Although the names of these additional fields indicate their purpose to a great extent, I will explain the usefulness of these global fields in later chapters as you use them.

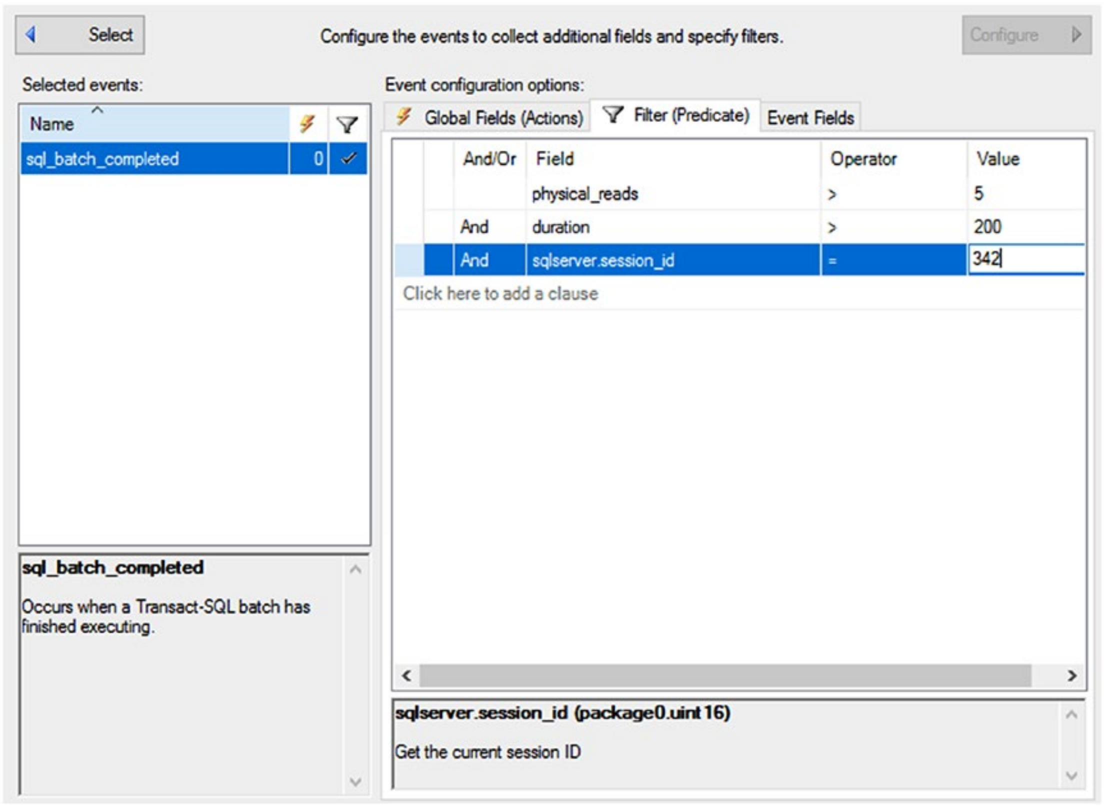
Event Filters

In addition to defining events and actions for an Extended Events session, you can define various filter criteria. These help keep the session output small, which is usually a good idea. You can add filters for event fields or global fields. You also get to choose whether you want each filter to be an OR or an AND to further control the methods of filtering. You can decide on the comparison operator, such as less than, equal to, and so on. Finally, you set a value for the comparison. All this will act to filter the events captured, reducing the amount of data you're dealing with and, possibly, the load on your system. Table 6-3 describes the filter criteria that you may commonly use during performance analysis.

Table 6-3. *SQL Trace Filters*

Events	Filter Criteria Example	Use
`sqlserver.username`	= <some value>	This captures events only for a single user or login.
`sqlserver.database_id`	= <ID of the database to monitor>	This filters out events generated by other databases. You can determine the ID of a database from its name as follows: `SELECT DB_ID('AdventureWorks20012')`.
`duration`	>= 200	For performance analysis, you will often capture a trace for a large workload. In a large trace, there will be many event logs with a duration that is less than what you're interested in. Filter out these event logs because there is hardly any scope for optimizing these SQL activities.
`physical_reads`	>= 2	This is similar to the criterion on the duration filter.
`sqlserver.session_id`	= <Database users to monitor>	This troubleshoots queries sent by a specific server session.

Figure 6-4 shows a snippet of the preceding filter criteria selection in the Session window.


Select
Configure the events to collect additional fields and specify filters.
Configure
Selected events:
Event configuration options:
Global Fields (Actions)
Filter (Predicate)
Event Fields
Click here to add a clause

Name		
sql_batch_completed	0	✓

And/Or	Field	Operator	Value
	physical_reads	>	5
And	duration	>	200
And	sqlserver.session_id	=	342

sql_batch_completed

Occurs when a Transact-SQL batch has finished executing.

sqlserver.session_id (package0.uint16)

Get the current session ID

Figure 6-4. *Filters applied in the Session window*

If you look at the Field value in Figure 6-4, you'll note that it says `sqlserver.session_id`. This is because different sets of data are available to you, and they are qualified by the type of data being referenced. In this case, I'm talking specifically about a `sqlserver.session_id`. But I could be referring to something from SQL OS or even the Extended Events package itself.

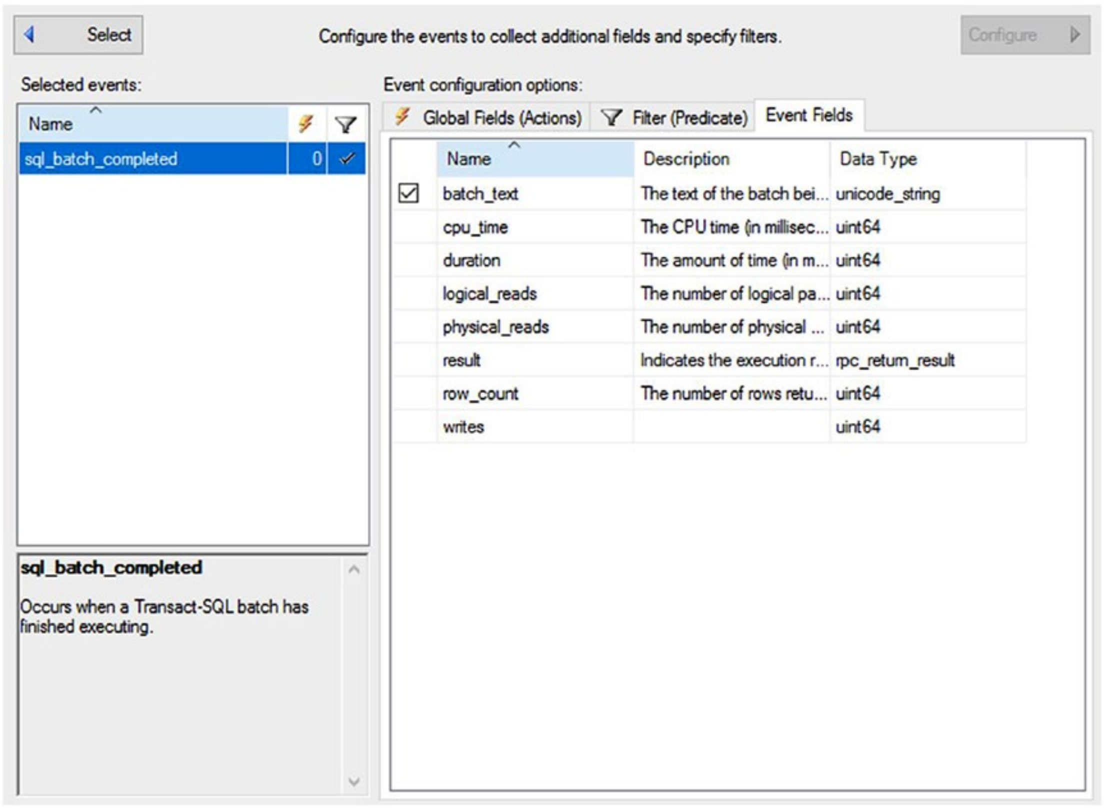
Event Fields

The standard event fields are included automatically with the event type. Table 6-4 shows some of the common actions that you use for performance analysis.

Table 6-4. *Actions Commands for Query Analysis*

Data Column	Description
`Statement`	The SQL text from the `rpc_completed` event.
`Batch_text`	The SQL text from the `sql_batch_completed` event.
`cpu_time`	The CPU cost of an event in microseconds (mc). For example, CPU = 100 for a `SELECT` statement indicates that the statement took 100mc to execute.
`logical_reads`	The number of logical reads performed for an event. For example, logical_reads = 800 for a `SELECT` statement indicates that the statement required a total of 800 page reads.
`Physical_reads`	The number of physical reads performed for an event. This can differ from the `logical_reads` value because of access to the disk subsystem.
`writes`	The number of logical writes performed for an event.
`duration`	The execution time of an event in ms.

Each logical read and write consists of an 8KB page activity in memory, which may require zero or more physical I/O operations. You can see the fields for any given event by clicking the Event Fields tab on display in Figure 6-5.


Select
Configure the events to collect additional fields and specify filters.
Configure
Selected events:
Event configuration options:
Global Fields (Actions)
Filter (Predicate)
Event Fields

Name		
sql_batch_completed	0	✓

sql_batch_completed

Occurs when a Transact-SQL batch has finished executing.

	Name	Description	Data Type
☑	batch_text	The text of the batch bei...	unicode_string
	cpu_time	The CPU time (in millisec...	uint64
	duration	The amount of time (in m...	uint64
	logical_reads	The number of logical pa...	uint64
	physical_reads	The number of physical ...	uint64
	result	Indicates the execution r...	rpc_return_result
	row_count	The number of rows retu...	uint64
	writes		uint64

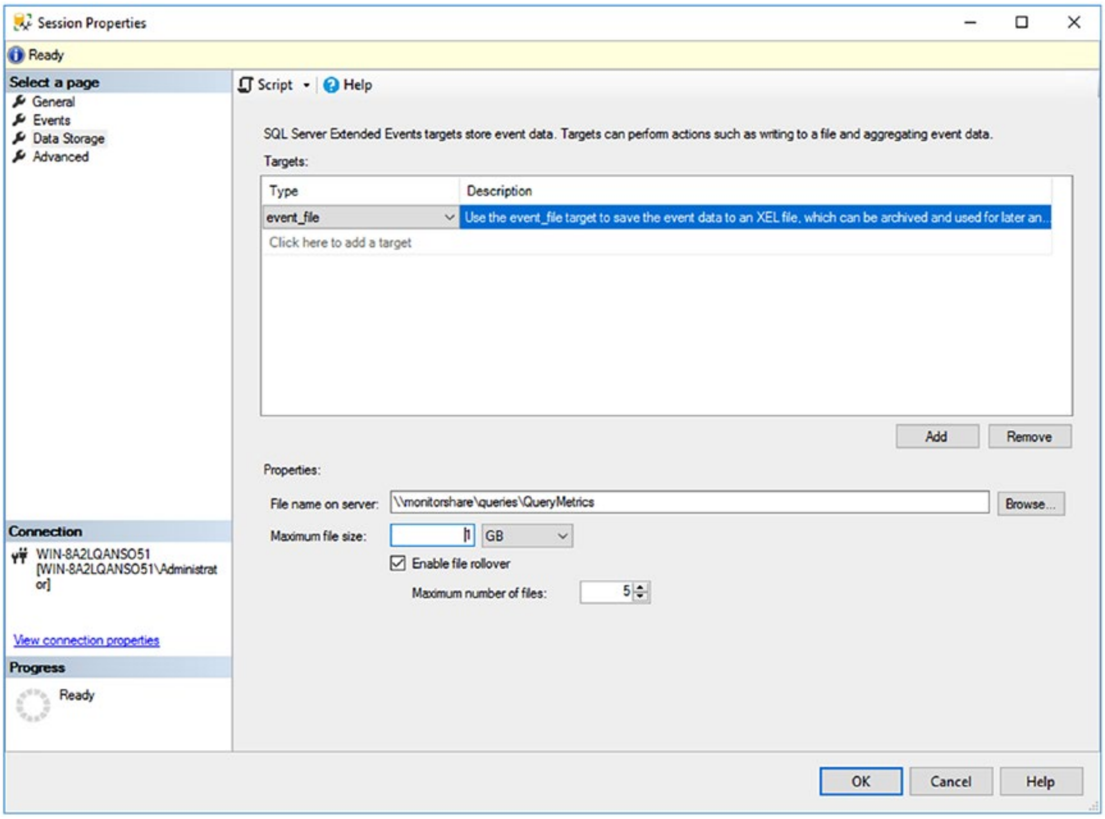
Figure 6-5. *New Session window with the Event Fields tab in Configure on display*

Some of the event fields are optional, but most of them are automatically included with the event. You can decide whether you want to include the optional fields. In Figure 6-5 you could exclude the `batch_text` field by clicking the check box next to it.

Data Storage

The next page in the new Session window, Data Storage in the "Select a page" pane, is for determining how you're going to deal with the data generated by the session. The output mechanism is referred to as the *target*. You have two basic choices: output the information to a file or simply use the buffer to capture the events. There are seven different types of output, but most of them are out of scope for the book. For the purposes of collecting performance information, you're going to use either `event_file` or `ring_buffer`. You should use only small data sets with the buffer because it will consume memory. Because it works with memory within the system, the buffer is built so that, rather than overwhelm the system memory, it will drop events, so you're more likely to lose information using the buffer. In most circumstances for monitoring query performance, you should capture the output of the session to a file.

You have to select your target, as shown in Figure 6-6.


Session Properties

Ready

SQL Server Extended Events targets store event data. Targets can perform actions such as writing to a file and aggregating event data.

Targets:

Type	Description
event_file	Use the event_file target to save the event data to an XEL file, which can be archived and used for later an...
Click here to add a target	

Properties:

File name on server: \\monitorshare\queries\QueryMetrics

Maximum file size: 1 GB

☑ Enable file rollover

Maximum number of files: 5

Connection: WIN-8A2LQANSO51 [WIN-8A2LQANSO51\Administrator]

Figure 6-6. *Data Storage window in the New Session window*

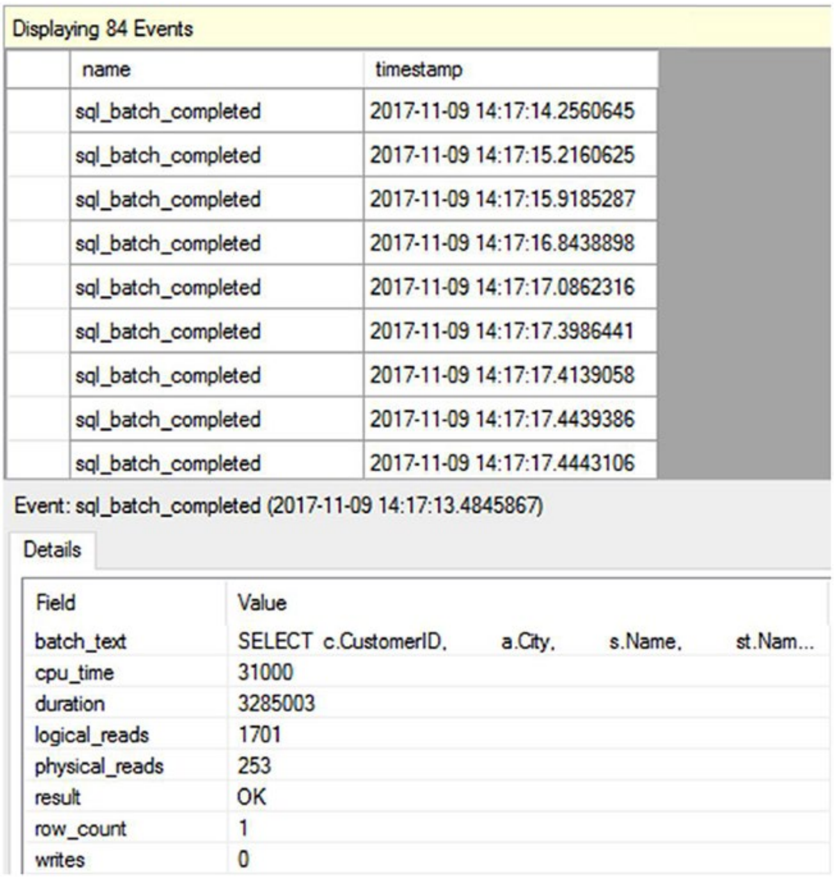
You should specify an appropriate storage location on your system. You can also decide whether you're using more than one file, how many, and whether those files roll over. All of those are management decisions that you'll have to deal with as part of working with your environment and your SQL query monitoring. You can run this 24/7, but you have to be prepared to deal with large amounts of data depending on how stringent the filters you've created are.

In addition to the buffer or the file, you have other output options, but they're usually reserved for special types of monitoring and not usually necessary for query performance tuning.

Finishing the Session

Once you've defined the storage, you've set everything needed for the session. There is an Advanced page as well, but you really shouldn't need to modify this from the defaults on most systems. When you click OK, the session will get created. If you instructed on the first tab that the session start after creation, it will start immediately, but whether it starts or not, it will be stored on the server. One of the beauties of Extended Events sessions is that they're stored on the server, so you can turn them on and off as needed with no need to re-create the session. The sessions are stored permanently until you remove them and will even survive a reboot, although, depending on how you've configured the session, you may have to restart them as necessary.

Assuming you either didn't automatically start the session or selected the option to watch the data live, you can do both to the session you just created. Right-click the session, and you'll see a menu of actions including Start Session, Stop Session, and Watch Live Data. If you start the session and you chose to observe the output, you should see a new window appear in Management Studio showing the events you're capturing. These events are coming off the same buffer as the one that is writing out to disk, so you can watch events in real time. Take a look at Figure 6-7 to see this in action.



Displaying 84 Events

name	timestamp
sql_batch_completed	2017-11-09 14:17:14.2560645
sql_batch_completed	2017-11-09 14:17:15.2160625
sql_batch_completed	2017-11-09 14:17:15.9185287
sql_batch_completed	2017-11-09 14:17:16.8438898
sql_batch_completed	2017-11-09 14:17:17.0862316
sql_batch_completed	2017-11-09 14:17:17.3986441
sql_batch_completed	2017-11-09 14:17:17.4139058
sql_batch_completed	2017-11-09 14:17:17.4439386
sql_batch_completed	2017-11-09 14:17:17.4443106

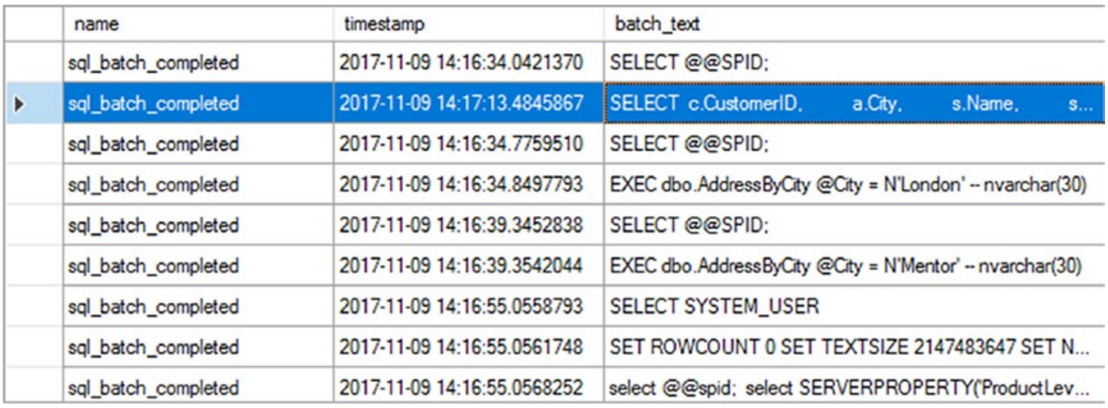
Event: sql_batch_completed (2017-11-09 14:17:13.4845867)

Details

Field	Value
batch_text	SELECT c.CustomerID, a.City, s.Name, st.Nam...
cpu_time	31000
duration	3285003
logical_reads	1701
physical_reads	253
result	OK
row_count	1
writes	0

Figure 6-7. *Live output of the Extended Events session created by the wizard*

You can see the events at the top of the window showing the type of event and the date and time of the event. Clicking the event at the top will open the fields that were captured with the event on the bottom of the screen. As you can see, all the information I've been talking about is available to you. Also, if you're unhappy with having a divided output, you can right-click a column and select Show Column in Table from the context menu. This will move it up into the top part of the screen, displaying all the information in a single location, as shown in Figure 6-8.



	name	timestamp	batch_text
	sql_batch_completed	2017-11-09 14:16:34.0421370	SELECT @@SPID;
▶	sql_batch_completed	2017-11-09 14:17:13.4845867	SELECT c.CustomerID, a.City, s.Name, s...
	sql_batch_completed	2017-11-09 14:16:34.7759510	SELECT @@SPID;
	sql_batch_completed	2017-11-09 14:16:34.8497793	EXEC dbo.AddressByCity @City = N'London' -- nvarchar(30)
	sql_batch_completed	2017-11-09 14:16:39.3452838	SELECT @@SPID;
	sql_batch_completed	2017-11-09 14:16:39.3542044	EXEC dbo.AddressByCity @City = N'Mentor' -- nvarchar(30)
	sql_batch_completed	2017-11-09 14:16:55.0558793	SELECT SYSTEM_USER
	sql_batch_completed	2017-11-09 14:16:55.0561748	SET ROWCOUNT 0 SET TEXTSIZE 2147483647 SET N...
	sql_batch_completed	2017-11-09 14:16:55.0568252	select @@spid; select SERVERPROPERTY('ProductLev...

Figure 6-8. *The statement column has been added to the table.*

You can also open the files you've collected through this interface and use it to browse the data. You can search within a column on the collected data, sort by them, and group by fields. One of the great ways to see an aggregate of all calls to a particular query is to use `query_hash`, a global field that you can add to your data collection. The GUI offers a lot of ways to manipulate the information you've collected.

Watching this information through the GUI and browsing through files is fine, but you're going to want to automate the creation of these sessions. That's what the next section covers.

The Built-in system_health Session

Built in to SQL Server and automatically running by default, there is an Extended Event session called `system_health`. It's primarily meant as a mechanism for observing the overall health of the system and collecting errors and diagnostics about internals. However, it also automatically captures some information that is useful when we're talking about query performance tuning.

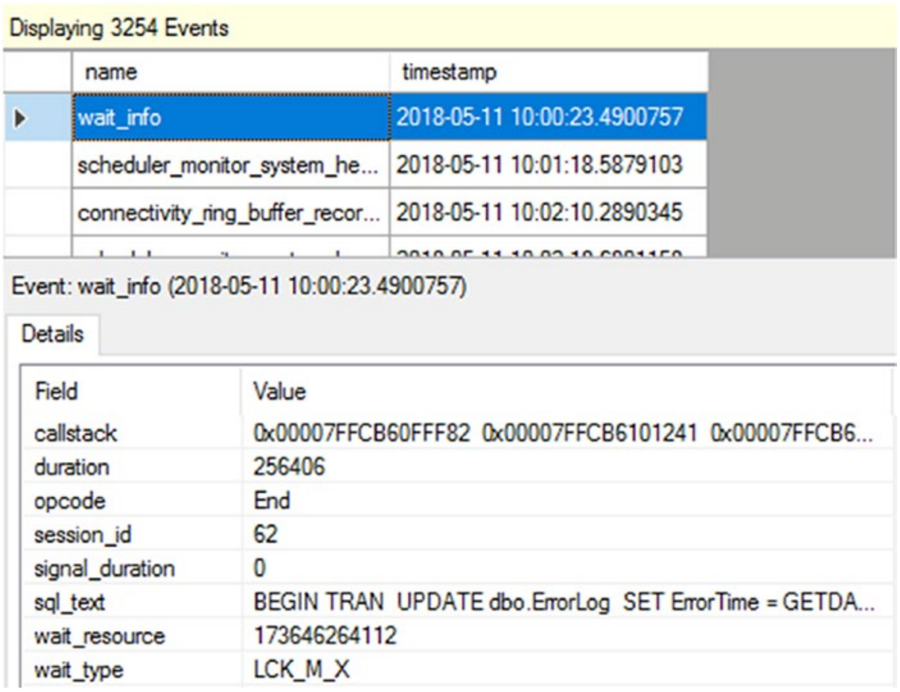
By default, out of the box, it collects the full information on deadlocks as they occur. Deadlocks are absolutely a performance issue and are covered in Chapter 22. The `system_health` Extended Events session means we don't have to do any other work to begin diagnosing deadlock situations.

The `system_health` session captures the `sql_text` and `session_id` for any processes that have waited on latches for longer than 15 seconds. That information is useful for immediately identifying queries that may need tuning. You also get the `sql_text` and `session_id` for any queries that waited longer than 30 seconds for a lock. Again, this is a way to identify immediately, with no other work than searching the `system_health` information, which queries may need tuning.

Because this is just another session, you have full control over it and can even remove it from your system, although I certainly don't recommend that. It collects its information in a 5MB file and keeps a rolling set of four files. You won't be able to go back to the beginning of your server install with this information, but it should have all the recent behavior of your server. The files are located by default with your other log files. You can find the location like this:

```
SELECT path
FROM sys.dm_os_server_diagnostics_log_configurations;
```

With that location you can query the session or open it in the Live Data explorer window, as shown in Figure 6-9.



Displaying 3254 Events

name	timestamp
wait_info	2018-05-11 10:00:23.4900757
scheduler_monitor_system_he...	2018-05-11 10:01:18.5879103
connectivity_ring_buffer_recor...	2018-05-11 10:02:10.2890345

Event: wait_info (2018-05-11 10:00:23.4900757)

Details

Field	Value
callstack	0x00007FFCB60FFF82 0x00007FFCB6101241 0x00007FFCB6...
duration	256406
opcode	End
session_id	62
signal_duration	0
sql_text	BEGIN TRAN UPDATE dbo.ErrorLog SET ErrorTime = GETDA...
wait_resource	173646264112
wait_type	LCK_M_X

Figure 6-9. *Wait_info event in the system_health Extended Event session*

The event on display in Figure 6-9 is the `wait_info` event, which shows that I had a process waiting to obtain a lock for more than 30 seconds. The `sql_text` field will show the query in question. As you can see, from a performance tuning standpoint, this is invaluable information. Best of all, it's available on your systems right now. You don't have to do anything to set it up.

Extended Events Automation

The ability to use the GUI to build a session and define the events you want to capture does make things simple, but, unfortunately, it's not a model that will scale. If you need to manage multiple servers where you're going to create sessions for capturing key query performance metrics, you're not going to want to connect to each one and go through the GUI to select the events, the output, and so on. This is especially true if you take into account the chance of a mistake. Instead, it's much better to learn how to work with sessions directly from T-SQL. This will enable you to build a session that can be run on a number of servers in your system. Even better, you're going to find that building sessions directly is easier in some ways than using the GUI, and you're going to be much more knowledgeable about how these processes work.

Creating a Session Script Using the GUI

You can create a scripted trace in one of two ways, manually or with the GUI. Until you get comfortable with all the requirements of the scripts, the easy way is to use the Extended Events GUI. These are the steps you'll need to perform:

1. Define a session.
2. Right-click the session, and select Script Sessions As, CREATE To, and File to output straight to a file. Or, use the Script button at the top of the New Session window to create a T-SQL command in the Query window.

These steps will generate the script you need to create a session and output it to a file. To manually create this new trace, use Management Studio as follows:

1. Open the script file or navigate to the Query window.
2. Modify the path and file location for the server you're creating this session on.
3. Execute the script.

Once the session is created, you can use the following command to start it:

```
ALTER EVENT SESSION QueryMetrics
ON SERVER
STATE = START;
```

You may want to automate the execution of the last step through the SQL Agent, or you can even run the script from the command line using the `sqlcmd.exe` utility. Whatever method you use, the final step will start the session. To stop the session, just run the same script with the STATE set to stop. I'll show how to do that in the next section.

Defining a Session Using T-SQL

If you followed the steps from the previous section to create a script, you would see something like this in your Query Editor window:

```
CREATE EVENT SESSION [QueryMetrics]
ON SERVER
    ADD EVENT sqlserver.sql_batch_completed
    (SET collect_batch_text = (1)
     WHERE ([sqlserver].[database_name] = N'AdventureWorks2017')
    )
    ADD TARGET package0.event_file
    (SET filename = N'q:\PerfData\QueryMetrics')
WITH
(
    MAX_MEMORY = 4096KB,
    EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS,
    MAX_DISPATCH_LATENCY = 30 SECONDS,
    MAX_EVENT_SIZE = 0KB,
    MEMORY_PARTITION_MODE = NONE,
    TRACK_CAUSALITY = OFF,
    STARTUP_STATE = OFF
);
GO
```

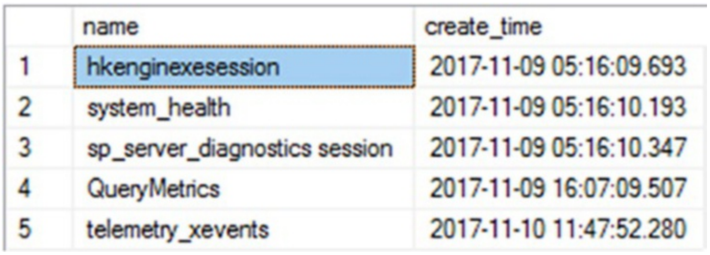
To create an Extended Events session, a single command defines the session, CREATE EVENT SESSION. You then just use ADD EVENT within that command to define the session. The filters are simply a WHERE clause added to each event definition. Finally, you add a target defining where the data captured should be stored. The WITH clause is actually just the default values from the Advanced page in the GUI. You can leave off the WITH clause and those values, and they'll still be set for the session.

Once the session has been defined, you can activate it using ALTER EVENT, as shown earlier.

Once a session is started on the server, you don't have to keep Management Studio or the Query Editor open anymore. You can identify the active sessions by using the dynamic management view sys.dm_xe_sessions, as shown in the following query:

```
SELECT  dxs.name,
        dxs.create_time
FROM    sys.dm_xe_sessions AS dxs;
```

Figure 6-10 shows the output of the view.



	name	create_time
1	hkenginexesession	2017-11-09 05:16:09.693
2	system_health	2017-11-09 05:16:10.193
3	sp_server_diagnostics session	2017-11-09 05:16:10.347
4	QueryMetrics	2017-11-09 16:07:09.507
5	telemetry_xevents	2017-11-10 11:47:52.280

Figure 6-10. *Output of sys.dm_xe_sessions*

The number of rows returned indicates the number of sessions active on SQL Server. I have four other sessions, all system defaults, running in addition to the one I created in this chapter. You can stop a specific session by executing the stored procedure ALTER EVENT SESSION.

```
ALTER EVENT SESSION QueryMetrics
ON SERVER
STATE = STOP;
```

To verify that the session is stopped successfully, reexecute the query against the catalog view `sys.dm_xe_sessions`, and ensure that the output of the view doesn't contain the named session.

Using a script to create your sessions allows you to automate across a large number of servers. Using the scripts to start and stop the sessions means you can control them through scheduled events such as through SQL Agent. In Chapter 20, you will learn how to control the schedule of a session while capturing the activities of a SQL workload over an extended period of time.

Note The time captured through a session defined as illustrated in this section is stored in microseconds, not milliseconds. This difference between units can cause confusion if not taken into account. You must filter based on microseconds.

Using Causality Tracking

Defining sessions through either the GUI or T-SQL is fairly simple. Consuming the information is also pretty easy. However, you'll quickly find that you don't simply want to observe single batch statements or single procedure calls. You're going to want to see all the statements within a procedure as well as the procedure call. You're going to want to see statement-level recompiles, waits, and all sorts of other events and have them all directly tied together back to an individual stored procedure or statement. That's where *causality tracking* comes in.

You can enable causality tracking as noted earlier through the GUI, or you can include it in an SQL command. The following script captures the start and stop of remote procedure calls and all the statements within those calls. I've also enabled causality tracking.

```
CREATE EVENT SESSION ProcedureMetrics
ON SERVER
    ADD EVENT sqlserver.rpc_completed
    (WHERE (sqlserver.database_name = N'AdventureWorks2017')),
    ADD EVENT sqlserver.rpc_starting
    (WHERE (sqlserver.database_name = N'AdventureWorks2017')),
    ADD EVENT sqlserver.sp_statement_completed
```