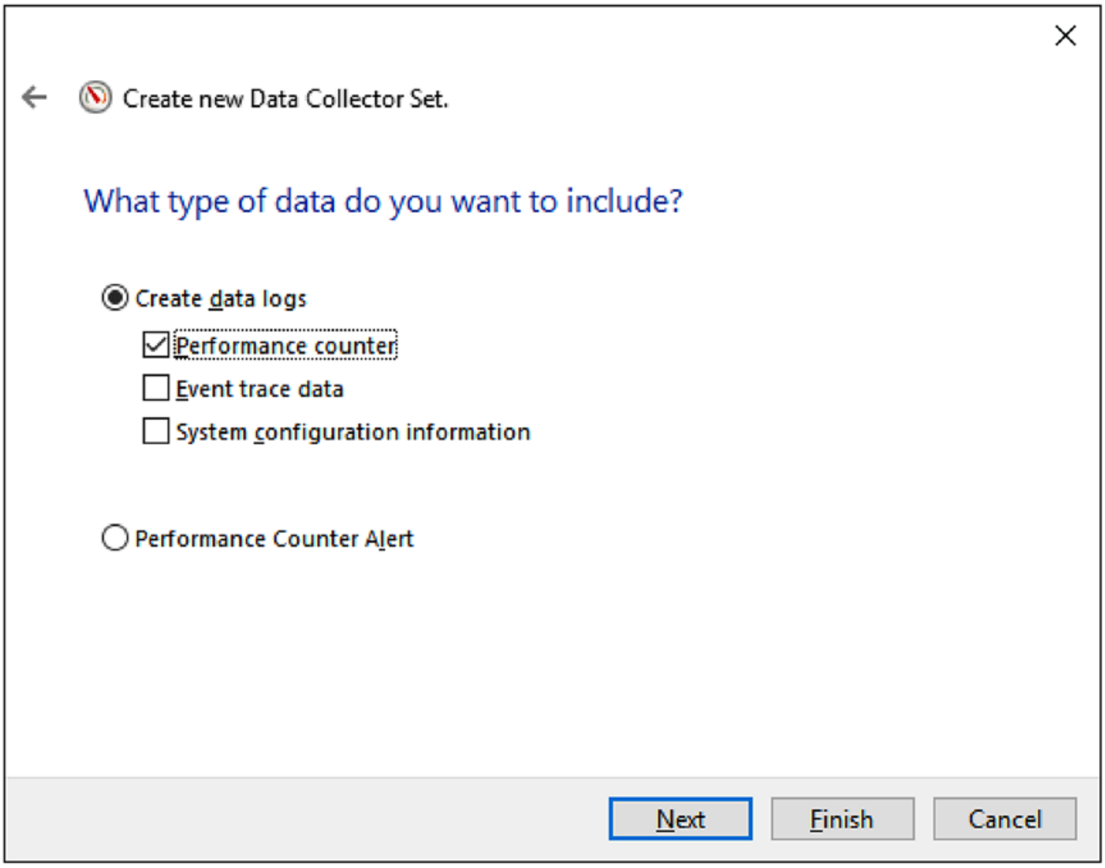

Create new Data Collector Set.

What type of data do you want to include?

[● Create data logs]

[✓ Performance counter]

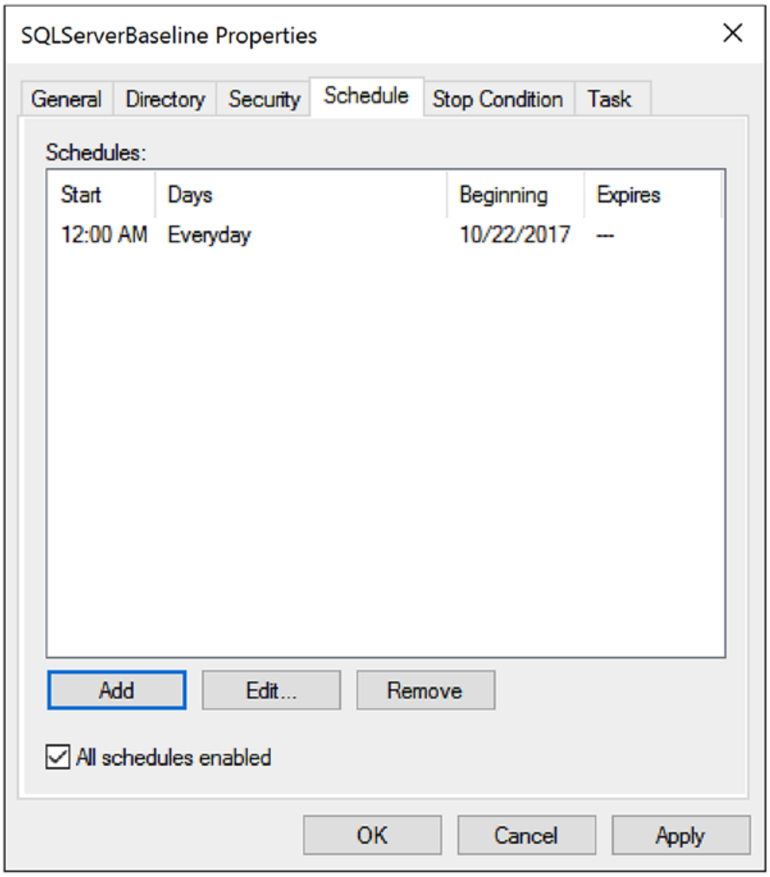
☐ Event trace data

☐ 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.



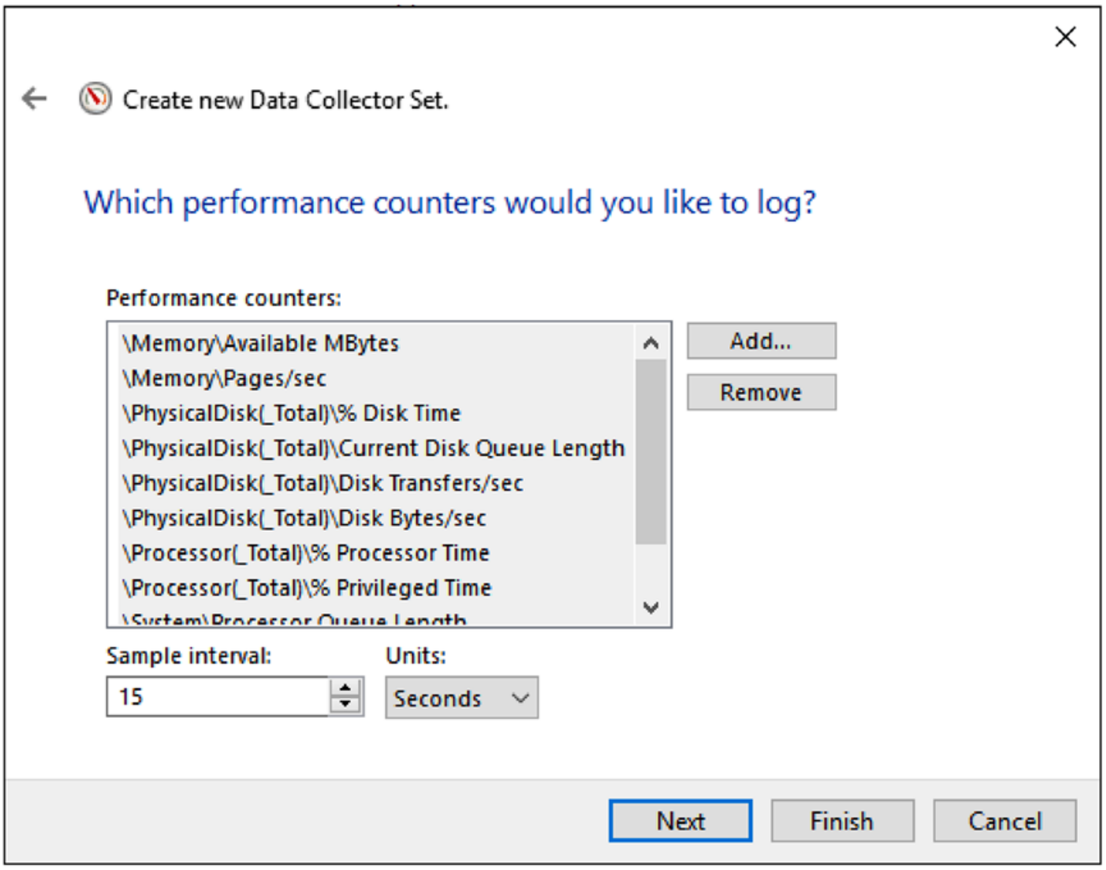
Schedules:

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

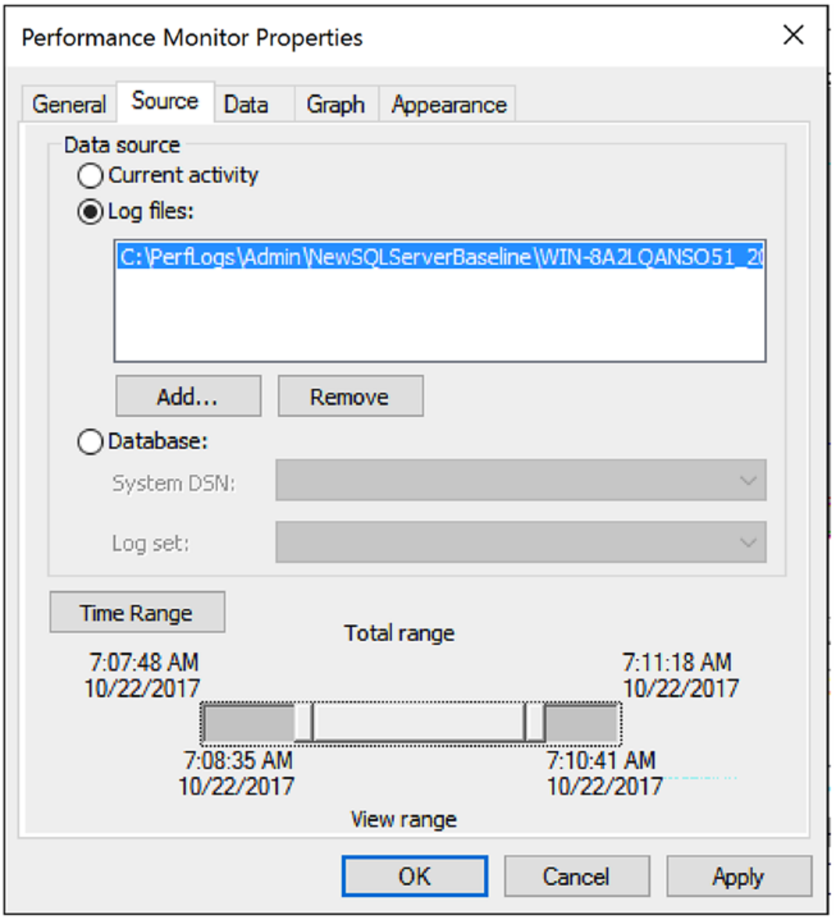
The default behavior of a database application changes over time because of various factors such as the following:

- 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

Data source
◯ Current activity
● Log files:

C:\PerfLogs\Admin\NewSQLServerBaseline\WIN-8A2LQANSO51_2(

◯ Database:
System DSN:
Log set:

Total range
7:07:48 AM 10/22/2017
7:11:18 AM 10/22/2017
7:08:35 AM 10/22/2017
7:10:41 AM 10/22/2017
View range

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).

© Grant Fritchey 2018
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.

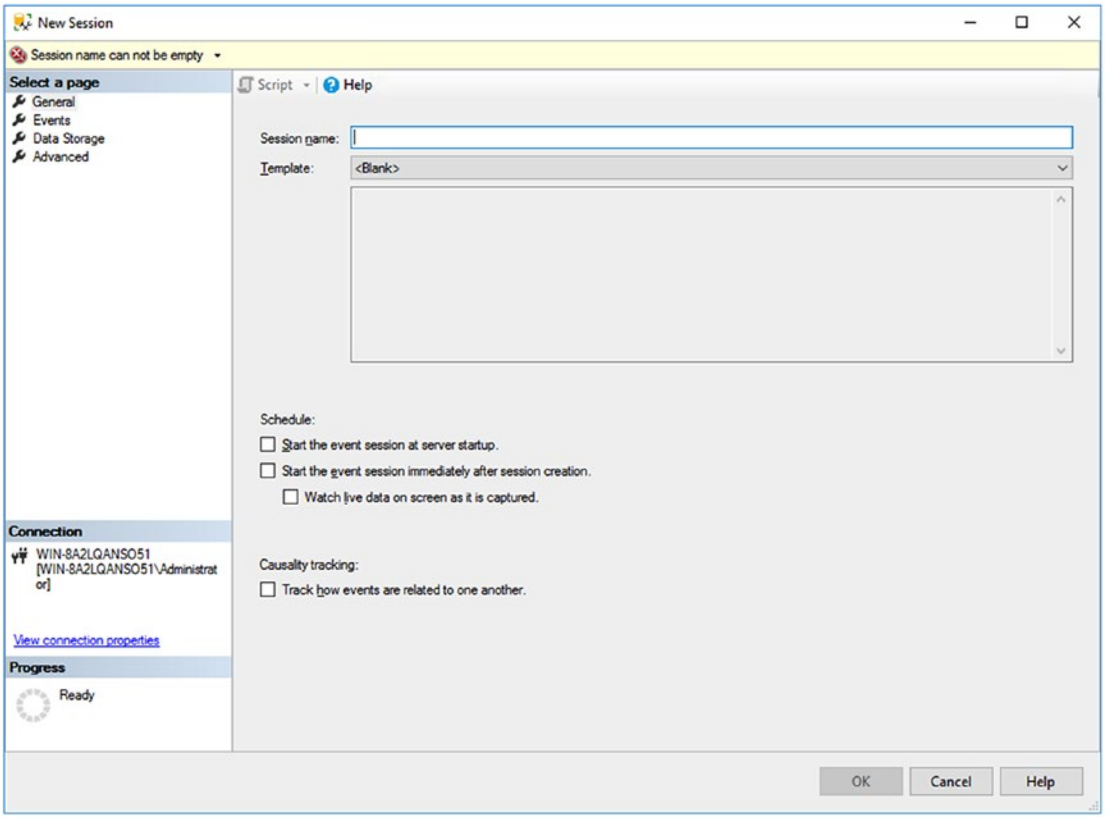

New Session
Session name can not be empty
Select a page
General
Events
Data Storage
Advanced
Script
Help
Session name:
Template:
<Blank>
Schedule:
Start the event session at server startup.
Start the event session immediately after session creation.
Watch live data on screen as it is captured.
Causality 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.

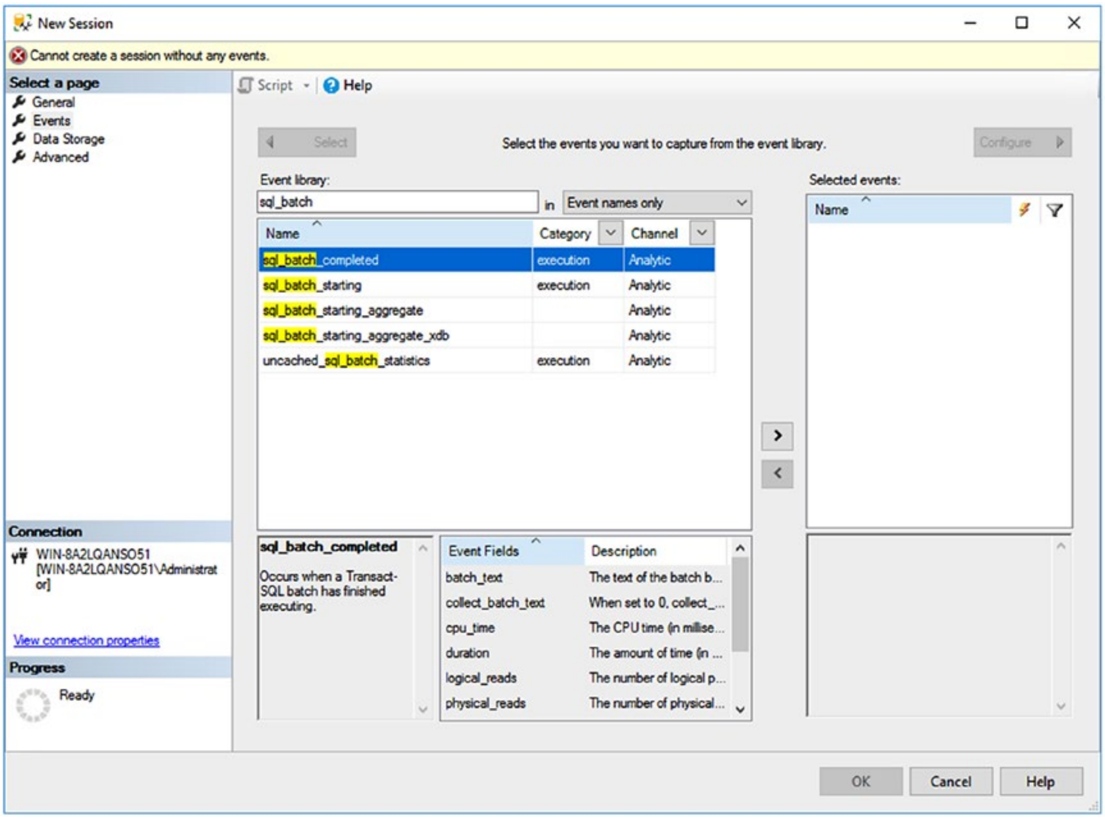
- *Query Detail Tracking*: This template is the same as Query Batch Tracking, but for every single statement in the system as well. This generates a large amount of data.
- *Query Wait Statistic*: This template captures wait statistics for each statement of every query and procedure for 20 percent of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


New Session

Cannot create a session without any events.

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

Script · Help

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]

View connection properties

Progress

Ready

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).

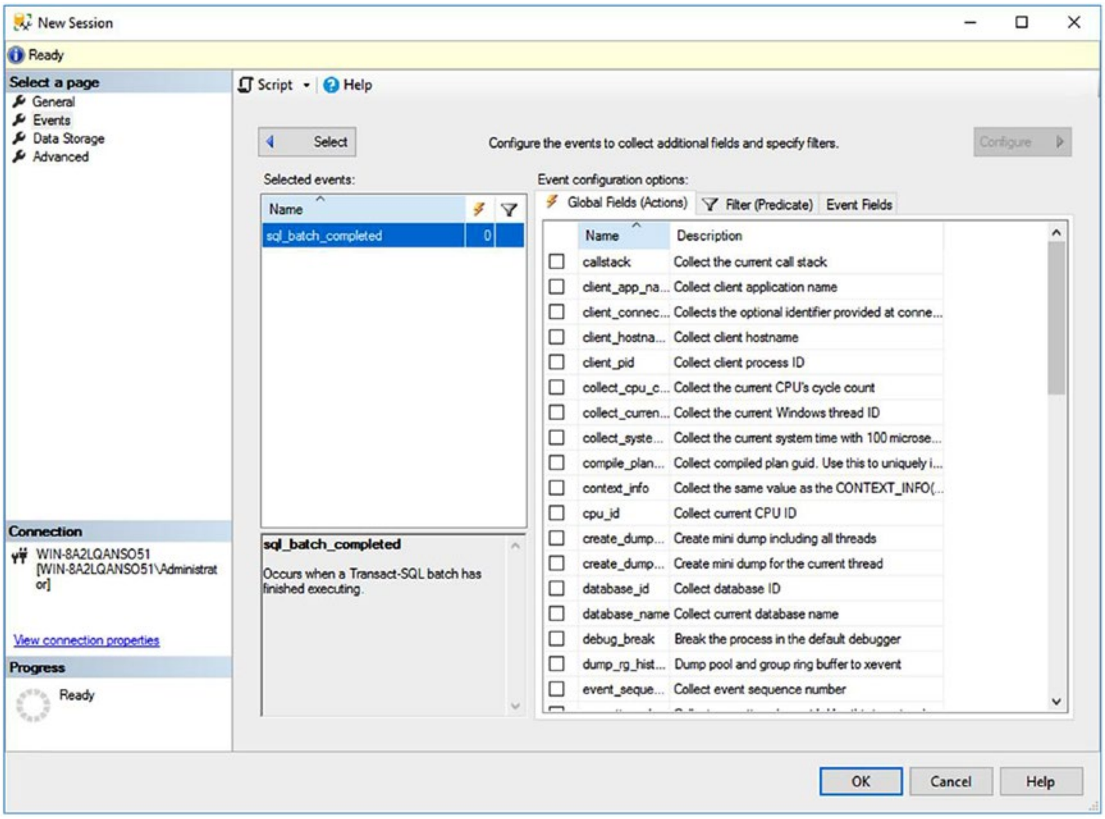
(*continued*)

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.



New Session

Ready

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

Script ▾ Help

Select

Configure the events to collect additional fields and specify filters.

Configure

Selected events:

Name	⚡	▽
sql_batch_completed	0	

sql_batch_completed

Occurs when a Transact-SQL batch has finished executing.

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

Connection

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

View connection properties

Progress

Ready

OK Cancel Help

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.