29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 787

CHAPTER 22

Performance Tuning and Troubleshooting SQL Server 2005

Performance and troubleshooting in SQL Server 2005 go hand in hand for database administrators. Most often, troubleshooting a SQL Server comes down to troubleshooting performance issues.

Performance tuning and troubleshooting performance issues are iterative processes consisting of monitoring, troubleshooting, adjusting, and repeating. When you're troubleshooting performance, it is helpful to have a goal in mind, such as having a specific query execute in less than *X* seconds.

Performance tuning and troubleshooting can be viewed as a layered model. The layers consist of the hardware, operating system, SQL Server, database, and application. Each layer is dependent on the layers below for its performance, as illustrated in Figure 22.1. For example, if the hardware layer is not performing due to a lack of resources, this affects the performance of the database layer.

It makes little or no sense to optimize the upper layers if the lower layers have not been optimized.

Note

Throughout the chapter, there are references to collection performance counters and various logs such as the SQL Server logs and Windows event logs. Although it is not covered in this chapter, Operations Manager 2007 is a great tool for collecting and keeping a long-term history of all the counters covered in this chapter. It is strongly recommended that you deploy and use Operations Manager 2007 to monitor the SQL Server 2005 infrastructure. See Chapter 21, "Monitoring SQL Server 2005" (online), to see how to use Operations Manager 2007 as well as the native monitoring tools.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 788

Appropriately, this chapter starts with troubleshooting performance at the lowest level.

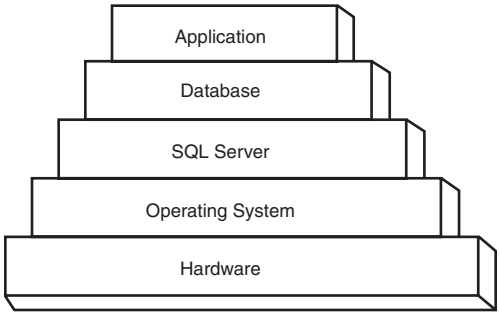

FIGURE 22.1
Optimization layers.

Platform Troubleshooting and Optimization

At the root of almost all optimizations are the hardware and operating system that SQL Server resides on, collectively referred to as the *platform*. These areas are likely to be the first place to look to optimize, thus ensuring that the SQL Server 2005 infrastructure rests on a solid platform.

Platform Performance Measures

If the hardware is not sufficient for the load placed on it by SQL Server, the result is performance issues and failures. When you're troubleshooting performance, this is one of the first areas to look into.

A set of key performance counters lets you know if the platform is experiencing performance problems. Capture these counters to understand how the hardware and operating system are performing at a high level. The objects and counters are

- **Memory: Pages/sec**—The Pages/sec counter exposes the rate at which pages are read or written to disk during hard page faults. As the memory pages are transferred to and from the relatively slow disk storage, the system will experience slow performance. The counter should be 20 or less on average, although it may spike. Add memory to the server if this number is too high.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 789

- **Memory: Available Bytes**—The Available Bytes counter exposes the amount of physical memory available for allocation. There should be at least 5MB of free RAM. If there is less than 5MB, consider adding RAM to the server.
- **Network Interface: Bytes Total/sec**—The Bytes Total/sec counter is the per second rate at which data is passing though the network interface card (NIC).
- **Physical Disk: % Disk Time**—The % Disk Time counter is the percent time that the disk is busy. This should be less than 55% over any sustained period of time, although it may spike occasionally. If this number is too high, consider adding drives to the array to increase the spindles and spread the load, adding additional channels, or changing the RAID version to a higher performance version (for example, RAID 5 to RAID 0+1).
- **Physical Disk: Avg. Disk Queue Length**—The Avg. Disk Queue Length counter exposes the number of disk requests that are waiting in the queue. According to queuing theory, this should be less than 2 over any sustained period of time or the queue could become backlogged. If this number is too high, consider adding drives to the array to increase the spindles and spread the load, adding additional channels, or changing the RAID version to a higher performance version (for example, RAID 5 to RAID 0+1).
- **Processor: % Processor Time**—The % Processor Time counter exposes the time the processor is doing actual work. This value is arrived at in a backward fashion by measuring the percentage of time the processor is idle and subtracting that from 100 to get the time the processor is busy doing work. This should be less than 80% over any sustained period of time, although it will spike occasionally. If this number is too high, consider adding or upgrading the processors on the server.
- **System: Processor Queue Length**—The Processor Queue Length counter exposes the number of threads in the processor queue. There is a single queue for processor time even on computers with multiple processors. If this exceeds 2 per processor (that is, 4 on a 2-processor server or 8 on a 4-processor server), consider adding or upgrading the processors on the server.

Record the data in a sheet similar to Table 22.1. Better yet, enter it into a spreadsheet or a database for long-term storage and a historical record.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 790

Table 22.1 **Hardware Counter Datasheet**

Counter Name	Minimum	Maximum	Average
Memory: Pages/sec			
Memory: Available Bytes			
Network Interface: Bytes Total/sec			
Physical Disk: % Disk Time			
Physical Disk: Avg. Disk Queue Length			
Processor: % Processor Time			
System: Processor Queue Length			

Capturing the Performance Measures

To illustrate the process, this section steps you through capturing and analyzing the performance counters. For ease of processing, the counters are captured to a counter log in the Performance tool.

To capture the counter logs, follow these steps:

1. Select Start, All Programs, Administrative Tools, Performance.
2. Expand the Performance Logs and Alerts folder.
3. Right-click on the Counter Logs and select New Log Settings.
4. Enter **`SQL Hardware Tuning`** for the name and click OK.
5. Click Add Counters and add all the counters in Table 22.1.
6. Click Close to close the window.
7. Click OK to save the settings and start the log.

Let the log capture data over a long enough period to capture a good profile of the utilization. Typically, this is at least 24 hours and not more than 7 days. You can stop the log collection either manually or on a schedule in the counter.

To view the logs, follow these steps:

1. Stop the counter log by right-clicking on it and selecting Stop.
2. Click on the System Monitor object.
3. Click on the View Log Data icon.
4. Select Log Files as the data source and add the log file from the counter log.
5. Click OK to show the graph of the counters in the log.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 791

The example in Figure 22.2 shows a log taken over 2 hours (2:09:05) for the recommended performance counters. The nice part of this log is that the tool summarizes the average, minimum, and maximum for each counter.

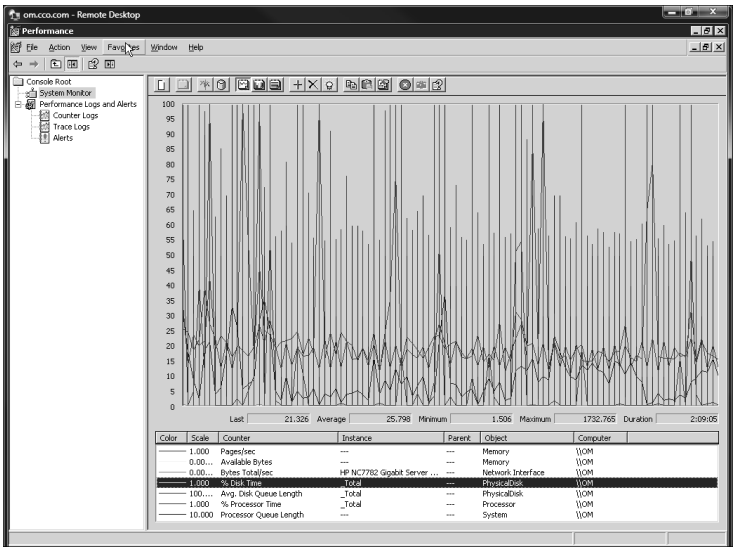

FIGURE 22.2
Performance log counters.

This information is recorded in Table 22.2, which is a summary of each of the counters.

Table 22.2 **Sample Counter Datasheet**

Counter Name	Minimum	Maximum	Average
Memory: Pages/sec	0.000	2663.928	9.338
Memory: Available Bytes	89804800	221859840	129313107
Network Interface: Bytes Total/sec	259.412	24480.983	2648.611
Physical Disk: % Disk Time	1.506	1732.765	25.798
Physical Disk: Avg. Disk Queue Length	0.015	17.328	0.258
Processor: % Processor Time	0.000	99.844	18.516
System: Processor Queue Length	0	17	1

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 792

Based on the previous description of the counters, the following recommendations can be made:

- **Memory: Pages/sec**—The Pages/sec counter is averaging 9.338 and is well under the limit of 20 with no sustained peaks.
- **Memory: Available Bytes**—Memory is above 89MB even at the minimum and is well above the 5MB guideline.
- **Network Interface: Bytes Total/sec**—This value peaking at 24KB is low utilization for a gigabit interface.
- **Physical Disk: % Disk Time**—The average of 25% with no sustained peaks is well within the guideline of 55% or less.
- **Physical Disk: Avg. Disk Queue Length**—The average of 0.258 is well under the guideline of 2.
- **Processor: % Processor Time**—The average of 18.516% is well within the guideline of less than 80% over sustained periods. The counter does peak frequently at 55%, but even that is still well within tolerances.
- **System: Processor Queue Length**—The system is a dual core, so the average of 1 is well within the guideline of 2 per core. There are frequent peaks to a max of 17, so it might be an area to monitor.

Overall, the sample log shows no hardware performance issues and thus no recommended upgrades, so the platform gets a clean bill of performance health.

Component Troubleshooting and Optimization

At the next level up in the performance diagram is SQL Server 2005. This server is broken into the Database Engine, Replication Services, Reporting Services, Analysis Services, and Integration Services components. These services are typically subject to performance issues that require troubleshooting.

Each SQL Server 2005 component requires a slightly different approach and different tools to troubleshoot performance.

Database Engine

There are a number of counter objects and counters that SQL Server 2005 exposes to monitor the SQL Server Database Engine.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 793

The primary counters for performance troubleshooting are

- **SQL Server:Buffer Manager: Buffer Cache Hit Ratio**—Specifies the percentage of pages found in the buffer cache without having to read from disk. The ratio is the total number of cache hits divided by the total number of cache lookups over the last few thousand page accesses.
- **SQL Server:General Statistics: User Connections**—Counts the number of users currently connected to SQL Server.

If the Buffer Cache Hit Ratio is too low, you need to install additional memory in SQL Server. The value should be at least 90% and ideally at 99%. However, with large databases with extensive access, this number might be difficult to achieve.

Although there is no specific guideline on the number of users shown by the User Connections counter, the utilization of SQL Server is proportional to the number of users. The more users the server is supporting, the higher the load on the server.

Other important areas to monitor when troubleshooting the SQL Server 2005 Database Engine are the Windows application log and SQL Server error logs.

The Windows Application Log contains application-level logs, including those from the SQL Server 2005 application. SQL Server and the SQL Server Agent both log events to the log. These logs should be reviewed when you're troubleshooting SQL Server 2005.

The SQL Server error log contains more detailed information than the Windows application logs and more pertinent to the SQL Server application. The SQL Log File Viewer supports viewing various logs at the same time, interleaving the log entries for easy correlation, as shown in Figure 22.3. The figure shows the aggregation of the SQL Agent log, SQL Server log, Windows application log, and Windows security log. This reduces the level of effort needed to troubleshoot problems because events that are related and causative can be seen in the same window.

You can access the SQL Server logs from the SQL Server Management Studio by selecting Management, SQL Server Logs.

Additional performance troubleshooting measures for the Database Engine are covered in the section "Database Troubleshooting and Optimization" later in this chapter.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 794

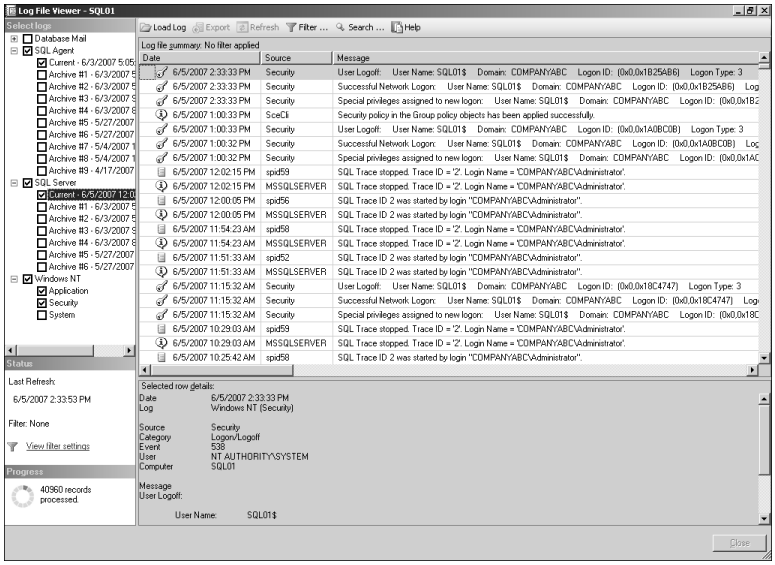

FIGURE 22.3
SQL Log File Viewer correlation.

Replication Services

Much of the replication troubleshooting and optimization is essentially the same as for the SQL Server Database Engine. However, the SQL Server Replication Monitor allows you to monitor the performance of replication.

The Replication Monitor shows information on

- Agent history
- Performance statistics
- Thresholds, warnings, and alerts

The agent history includes the messages from the Snapshot Agent, Log Reader Agent, and Queue Reader Agent. The messages show information, warnings, and alerts.

The performance statistics include information on the transactional and merge replication current performance. For transactional replication, latency is the critical measure. Table 22.3 shows the relationship between the value and latency threshold. The value depends on the latency compared to the

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 795

threshold; that is, a lower number is better. If the latency is 10 seconds and the threshold is 60 seconds, the latency is 15% of the threshold and the column value shows Excellent.

Table 22.3 **Transactional Replication Latency Performance**

Excellent	Good	Fair	Poor	Critical
0–34%	35–59%	60–84%	85–99%	>100%

For merge replication, performance is rated by comparing the row synchronization history of all subscriptions in the publication with the same connection type. The performance measure is expressed as a percentage of the merge subscription compared to the average. A higher number is better. If the publication subscriber average is 50 rows per second and a subscription is synchronizing at 25 rows per second, the percentage is 50% and the performance value shows Fair. Table 22.4 shows the values.

Table 22.4 **Merge Replication Performance**

Excellent	Good	Fair	Poor
151+%	76–150%	26–75%	0–25%

These values can be used to gauge how well replication is performing.

Finally, threshold, warnings, and alerts can be set to notify of performance issues. This capability is useful for troubleshooting performance issues, in that it keeps a record of time when performance drops. The thresholds can generate warnings and alerts if there are

- Impending subscription expirations
- Latency issues (transaction replication)
- Synchronization delays (merge replications)
- Subpar performance (merge replications)

All of these help you troubleshoot performance issues with Replication Services.

Reporting Services

Reporting Services performance troubleshooting is fundamentally based on the Database Engine and Windows Internet Information Services (IIS).

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 796

In addition to the database tools, the tools used to troubleshoot performance issues with Reporting Services are

- Windows Event Viewer
- Performance Monitor

Because much of the Reporting Services depends on the web services and Database Engine, performance problems can be addressed in these areas.

There are two Reporting Services–specific objects in the Performance Monitor:

- MSRS 2005 Web Service, which monitors report server performance
- MSRS 2005 Windows Service, which monitors scheduled operations and report delivery

In the objects, MSRS is the acronym for Microsoft Reporting Services. The MSRS 2005 Web Service performance object includes a collection of counters used to track report server processing of interactive report viewing operations. These counters are reset whenever ASP.NET stops the Reporting Services Web Service. Table 22.5 lists the MSRS 2005 Web Service counters.

Table 22.5 **MSRS 2005 Web Service Performance Counters**

Counter	Description
Active Sessions	Number of active sessions.
Cache Hits/Sec	Number of requests per second for cached reports.
Cache Misses/Sec	Number of requests per second that failed to return a report from cache.
First Session Requests/Sec	Number of new user sessions that are started from the report server cache each second.
Memory Cache Hits/Sec	Number of times per second that reports are retrieved from the in-memory cache.
Memory Cache Misses/Sec	Number of times per second that reports could not be retrieved from the in-memory cache.
Next Session Requests/Sec	Number of requests per second for reports that are open in an existing session.
Report Requests	Number of reports that are currently active and being handled by the report server.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 797

Table 22.5 **continued**

Counter	Description
Reports Executed/Sec	Number of successful report executions per second.
Requests/Sec	Number of requests per second made to the report server.
Total Cache Hits	Total number of requests for reports from the cache after the service started.
Total Cache Misses	Total number of times that a report could not be returned from the cache after the service started.
Total Memory Cache Hits	Total number of cached reports returned from the in-memory cache after the service started.
Total Memory Cache Misses	Total number of cache misses against the in-memory cache since the service started.
Total Processing Failures	Total number of report processing failures that have occurred since the service started.
Total Rejected Threads	Total number of data processing threads rejected for asynchronous processing and subsequently handled as synchronous processes in the same thread.
Total Reports Executed	Total number of reports that ran successfully after the service started.
Total Requests	Total number of all requests made to the report server after the service started.

The MSRS 2005 Windows Service performance object includes a collection of counters used to track report processing that is initiated through scheduled operations. Scheduled operations include subscription and delivery, report execution snapshots, and report history. The MSRS 2005 Windows Service performance counters are shown in Table 22.6.

Table 22.6 **MSRS 2005 Windows Service Performance Counters**

Counter	Description
Active Sessions	Number of active sessions stored in the report server database.
Cache Flushes/Sec	Number of cache flushes per second.
Cache Hits/Sec	Number of requests per second for cached reports.
Cache Misses/Sec	Number of requests per second that failed to return a report from cache.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 798

Table 22.6 **continued**

Counter	Description
Delivers/Sec	Number of report deliveries per second, from any delivery extension.
Events/Sec	Number of events processed per second.
Memory Cache Hits/Sec	Number of times per second that reports are retrieved from the in-memory cache.
Memory Cache Misses/Sec	Number of times per second that reports cannot be retrieved from the in-memory cache.
Next Session Requests/Sec	Number of requests per second for reports that are open in an existing session.
Report Requests	Number of reports that are currently active and being handled by the report server.
Reports Executed/Sec	Number of reports successfully generated per second.
Total App Domain Recycles	Total number of application domain cycles after the Report Server Windows Service started.
Total Cache Flushes	Total number of report server cache updates after the service started.
Total Cache Hits	Total number of requests for reports processed directly from the cache after the Report Server Windows Service started.
Total Cache Misses	Total number of times that a report could not be returned from cache after the Report Server Windows Service started.
Total Deliveries	Total number of reports delivered by the Scheduling and Delivery Processor, for all delivery extensions.
Total Events	Total number of events after the Report Server Windows Service started.
Total Memory Cache Hits	Total number of cached reports returned from the in-memory cache after the Report Server Windows Service started.
Total Memory Cache Misses	Total number of cache misses against the in-memory cache after the service started.
Total Processing Failures	Total number of report processing failures that have occurred since the service started.
Total Rejected Threads	Total number of data processing threads rejected for asynchronous processing and subsequently handled as a synchronous process in the same thread.
Total Reports Executed	Total number of reports run.
Total Requests	Total number of reports that ran successfully after the service started.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 799

Both of these sets of counters can be used to monitor and troubleshoot performance issues with Reporting Services.

Analysis Services

Analysis Services can be resource intensive, especially for memory. Memory is used during querying to speed up data retrieval, specifically caching data results, calculation results, and dimension data. Memory is also used during processing to store, index, and aggregate data temporarily. If not enough memory is available, the analysis jobs can become blocked and kill performance.

Analysis Services can use up to 3GB of memory on a 32-bit platform, but is not limited to 3GB on a 64-bit platform. For intensive Analysis Services applications, it is important to use 64-bit SQL Server 2005.

Memory allocation in Analysis Services is controlled by two memory parameters:

- **Memory\TotalMemoryLimit**—The upper limit percentage of the total physical memory in the server that Analysis Services will use. This defaults to 80%, that is, 80% of the physical memory of the server.
- **Memory\LowMemoryLimit**—The lower limit percentage of the total physical memory in the server that Analysis Services will use.

These settings can be found on the General page of the properties of the Analysis Services server in SQL Server Management Studio.

These memory settings are used to guide how Analysis Services manages memory under the following memory conditions:

- **No Pressure**—If the current Analysis Services memory usage is below the LowMemoryLimit, the cleaner does not free memory.
- **Some Pressure**—If the current Analysis Services memory usage is above the LowMemoryLimit but below the TotalMemoryLimit, the cleaner starts to free memory where it will not affect performance.
- **High Pressure**—If the current Analysis Services memory usage is above the TotalMemoryLimit, the cleaner aggressively frees memory even if it will affect performance.

Given the potential impact on performance that this memory management can have, it is important to monitor this when conducting performance troubleshooting. Table 22.7 lists the counters that you can use to troubleshoot memory performance issues with Analysis Services.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 800

Table 22.7 **Analysis Services Memory Counters**

Counter	Description
MSAS 2005:Memory\Memory Limit Low KB	Displays the Memory\LowMemoryLimit setting in Analysis Services. The default is 75.
MSAS 2005:Memory\Memory Limit High KB	Displays the Memory\TotalMemoryLimit setting in Analysis Services. The default is 80.
MSAS 2005:Memory\Memory Usage KB	Displays the memory usage of the server process. This is the value that is compared to Memory\LowMemoryLimit and Memory\TotalMemoryLimit.
MSAS 2005:Memory\Cleaner Balance/sec	Shows how many times the current memory usage is compared against the settings. Memory usage is checked every 500ms, so the counter trends toward 2 with slight deviations when the system is under high stress.
MSAS 2005:Memory\Cleaner Memory nonshrinkable KB	Displays the amount of memory, in kilobytes, not subject to purging by the cleaner.
MSAS 2005:Memory\Cleaner Memory shrinkable KB	Displays the amount of memory, in kilobytes, subject to purging by the cleaner.
MSAS 2005:Memory\Cleaner Memory KB	Displays the amount of memory, in kilobytes, known to the background cleaner.

In most cases, it is critical not to allow Analysis Services to exceed the TotalMemoryLimit. If it does this consistently, consider adding more memory to the server, moving the server to a 64-bit platform, or reducing the load.

Integration Services

Integration Services fundamentally depends on the performance of the queries; hence, optimizing the queries and database access is critical to the performance of Integration Services.

Review the Progress tab information to see where the longest-running components are and then optimize them. This information can also be captured in the package and data flow logs.

Integration Services also includes a number of performance counters in the SQL Server:SSIS Pipeline object that can be used to troubleshoot performance problems. These counters are listed in Table 22.8.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 801

Table 22.8 **Integration Services Performance Counters**

Counter	Description
BLOB Bytes Read	The number of bytes of binary large object (BLOB) data that the data flow engine has read from all sources.
BLOB Bytes Written	The number of bytes of BLOB data that the data flow engine has written to all destinations.
BLOB Files in Use	The number of BLOB files that the data flow engine currently is using for spooling.
Buffer Memory	The amount of memory in use. This may include both physical and virtual memory. When this number is larger than the amount of physical memory, the Buffers Spooled count rises as an indication that memory swapping is increasing. Increased memory swapping slows performance of the data flow engine.
Buffers in Use	The number of buffer objects of all types that all data flow components and the data flow engine are currently using.
Buffers Spooled	The number of buffers currently written to the disk. If the data flow engine runs low on physical memory, buffers not currently used are written to disk and then reloaded when needed.
Flat Buffer Memory	The total amount of memory, in bytes, that all flat buffers use. Flat buffers are blocks of memory that a component uses to store data. A flat buffer is a large block of bytes that is accessed byte by byte.
Flat Buffers in Use	The number of flat buffers that the data flow engine uses. All flat buffers are private buffers.
Private Buffer Memory	The total amount of memory in use by all private buffers. A buffer is not private if the data flow engine creates it to support data flow. A private buffer is a buffer that a transformation uses for temporary work only. For example, the Aggregation transformation uses private buffers to do its work.
Private Buffers in Use	The number of buffers that transformations use.
Rows Read	The number of rows that a source produces. The number does not include rows read from reference tables by the Lookup transformation.
Rows Written	The number of rows offered to a destination. The number does not reflect rows written to the destination data store.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 802

When you're troubleshooting a package executing within Integration Services, use event handlers to troubleshoot package execution problems.

One of the most resource-intensive and performance-impacting operations is sorting within a package flow. This consumes large quantities of memory and processing resources. This is true if the package contains either the Sort transformation or a query within the data flow that includes the ORDER BY clause. The `IsSorted` hint property can be used to indicate to down-level components that the data is already sorted and bypass the sorting overhead.

Database Troubleshooting and Optimization

Database tuning can result in a tremendous boost in performance and address the root cause of many troubleshooting issues. When you're setting up a database, it can be difficult to know exactly what indexes to create because it may not be completely clear how applications will use the database. Database administrators rarely have insight on the types of queries and data access patterns an application will present to the database. On the other end, application developers rarely have any understanding about the inner workings of a database and may not even know precisely what their application is doing from a data perspective. Given the complexity of the situation, developing a fully tuned database for any given application straight out of development is extremely difficult.

Fortunately, it is possible to cut the Gordian knot by capturing the application behavior during actual use and then using that captured data to make tuning recommendations. There are two specialized tools to do this: SQL Server Profiler and the Database Engine Tuning Advisor.

The following sections conduct a basic optimization walkthrough to show how you can use the tools to optimize a database.

SQL Server Profiler

The SQL Server Profiler tool captures SQL Server 2005 events as they are generated on a SQL Server. The captured information, referred to as a *workload*, can be reviewed in the UI or saved to a trace file. The workload can be used to analyze performance or can be replayed to conduct N+1 testing. In N+1 testing, the workload would be replayed and the results analyzed. Adjustments would be made to the system, then the workload would be replayed and the results analyzed again. This is repeated N times until finally all issues are resolved in the final N+1 time.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 803

You can use the tool to

- Step through queries to troubleshoot problems
- Identify and optimize slow queries
- Capture traces for replay or analysis
- Conduct security audits of database activity
- Provide input to the Database Engine Tuning Advisor for database optimization

The SQL Server Profiler is invaluable for getting detailed insight into the internal workings of applications and databases from a real-world and real-time perspective.

Profiler Trace Templates

The Profiler tool can capture a wide variety of different event classes and data columns in the trace. They are easily specified in the trace templates.

The different default trace templates are shown in Table 22.9. Additional templates can be created for specific needs.

Table 22.9 **Default SQL Server Profiler Trace Templates**

Template	Template Purpose	Event Classes
Tuning	Captures information about stored procedures and Transact-SQL batch execution. Use to produce trace output that the Database Engine Tuning Advisor can use as a workload to tune databases.	RPC:Completed SP:StmtCompleted SQL:BatchCompleted
Standard	Generic starting point for creating a trace. Captures all stored procedures and TSQL batches that are run. Use to monitor general database server activity.	Audit Login Audit Logout ExistingConnection RPC:Completed SQL:BatchCompleted SQL:BatchStarting
SP_Counts	Captures stored procedure execution behavior over time.	SP:Starting
TSQL	Captures all TSQL statements that are submitted to SQL Server by clients and the time issued. Use to debug client applications.	Audit Login Audit Logout ExistingConnection RPC:Starting SQL:BatchStarting

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 804

Table 22.9 **continued**

Template	Template Purpose	Event Classes
TSQL_Duration	Captures all TSQL statements submitted to SQL Server by clients, their execution time (in milliseconds), and groups them by duration. Use to identify slow queries.	RPC:Completed SQL:BatchCompleted
TSQL_Grouped	Captures all TSQL statements submitted to SQL Server and the time they were issued. Groups information by user or client that submitted the statement. Use to investigate queries from a particular client or user.	Audit Login Audit Logout ExistingConnection RPC:Starting SQL:BatchStarting
TSQL_Replay	Captures detailed information about TSQL statements that is required if the trace will be replayed. Use to perform iterative tuning, such as benchmark testing.	CursorClose CursorExecute CursorOpen CursorPrepare CursorUnprepare Audit Login Audit Logout Existing Connection RPC Output Parameter RPC:Completed RPC:Starting Exec Prepared SQL Prepare SQL SQL:BatchCompleted SQL:BatchStarting
TSQL_SPs	Captures detailed information about all executing stored procedures. Use to analyze the component steps of stored procedures.	Audit Login Audit Logout ExistingConnection RPC:Starting SP:Completed SP:Starting SP:StmtStarting SQL:BatchStarting

You will most often use the Tuning template. It captures the events and columns that are used by the Database Engine Tuning Advisor.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 805

Database Engine Tuning Advisor

The Database Engine Tuning Advisor automates the process of selecting an optimized set of indexes, indexed views, statistics, and partitions and even provides the code to implement the recommendations it makes. To make your life even easier, you can use the tool to implement recommendations right from within the SQL Server Management Studio console.

The Database Engine Tuning Advisor can work with a specific query or can use a real-world workload as gathered by the SQL Server Profiler. The advantage of the latter approach is that the workload is generated based on actual usage, and the tuning recommendation reflects that.

The Database Engine Tuning Advisor is customizable and allows you to select the level of recommendation that the tool will recommend. This way, you can maintain the existing database design and make appropriate fine-tuning recommendations for just indexes. Or you can make the existing design flexible and then have the tool recommend far-reaching changes to the structure such as partitioning.

The following sections walk you through running the Database Engine Tuning Advisor, starting with capturing a workload.

Capturing a Workload

The first part of the process to run the SQL Server Profiler is to capture a workload. This example uses the Customer database from Chapter 14, "Encrypting SQL Server Data and Communications." See that chapter for the steps to create the Customer database.

To capture a workload, follow these steps:

1. Launch SQL Server Management Studio.
2. Open a query window by right-clicking on the server in Object Explorer and selecting New Query.
3. Select Start, All Programs, Microsoft SQL Server 2005, Performance Tools, SQL Server Profiler.
4. Select File, New Trace.
5. Connect to the database engine, in this case SQL01.
6. On the General tab of the Trace Properties, enter **`Customer Database Trace`** for the Trace name.
7. Select the Tuning template from the Use the Template pull-down.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 806

8. Check the Save to File box and select a location for the trace file.
9. Change the maximum file size to 100, although this example will not need this much space.
10. Select the Events Selection tab.
11. Review the events and columns that are preselected by the template you chose, in this case the Tuning template. Other templates select other events and columns.
12. Click Run to start the trace.

The trace window shows the server activity. By selecting any line in the trace window, you can review the event class, the duration of the statement (in milliseconds), the name of the database, the login name of the executing process, and even the detailed statement itself in the details window.

Now that the SQL Server Profiler is tracing the events on the server, a workload needs to be generated. Usually, you would do this during normal operations, and the trace would gather a workload. However, in this example, there are no normal operations. A series of query statements is executed to simulate a workload.

Return to the SQL Server Management Studio tool, which was left open at the query window. Then execute the following series of queries.

The first statement selects all columns from the database:

```
USE Customer;
GO
SELECT * FROM dbo.Customers;
GO
```

The first statement returns too many columns, so the following query narrows the data to just the columns needed:

```
SELECT FirstName, LastName, EmailAddress, Occupation, State FROM
➥dbo.Customers;
GO
```

This statement still returns too many rows, so the following query returns just the rows of management:

```
SELECT FirstName, LastName, EmailAddress, Occupation, State FROM
➥dbo.Customers
       WHERE Occupation = 'Management';
GO
```

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 807

However, the rows needed are just for California rather than all states, so the following query returns exactly what is needed:

```
SELECT FirstName, LastName, EmailAddress, Occupation, State FROM
➥dbo.Customers
       WHERE Occupation = 'Management' and State = 'CA';
GO
```

This final query should return 192 rows. This is a simple workload, but effective for a demonstration of the process.

The workload has been generated, so the next step is to stop the Profiler tool and save the workload for analysis. Follow these steps:

1. Switch to the SQL Server Profiler tool.
2. Select File, Stop Trace to stop the trace.
3. Scroll through the events and locate each of the query statements that you just executed. Note the duration, database, login name for each statement, and query for each event.
4. Close the SQL Server Profiler.

Now the workload has been saved and is ready for analysis.

Analyzing the Workload

Run the Tuning Advisor to analyze the workload as follows:

1. Select Start, All Programs, Microsoft SQL Server 2005, Performance Tools, Database Engine Tuning Advisor.
2. Connect to the Database Engine, in this case SQL01.
3. In the Workload section on the General tab, select the file that the trace was saved to, in this case `Customer Database Trace.trc`.
4. Select the Customer database from the Database for Workload Analysis pull-down.
5. In the Select Databases and Tables to Tune section, check the Customer database. The configuration should look similar to that in Figure 22.4.
6. Select the Tuning Options tab. There are a number of tuning options for the advisor to use. The advisor, by default, recommends index and statistic changes but can recommend changes to the physical design structures as well. Leave these at the default, which is to recommend index changes only.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 808

7. Select Actions, Start Analysis to begin the analysis. In a real-world situation, the workload would be much longer, and an analysis would take a significant amount of time. In the case of this simple simulation, the analysis will take less than a minute.

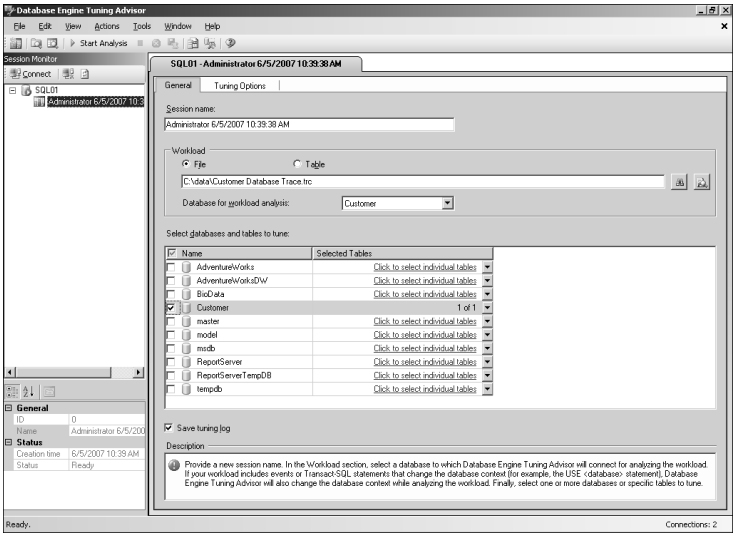

FIGURE 22.4
Database Engine Tuning Advisor settings.

Review the Results

After the analysis, the Database Engine Tuning Advisor shows the Recommendations tab of the tool, which offers a set of recommendations for the Customers database based on the workload. This view is shown in Figure 22.5.

The top line of the recommendations window shows an Estimated Improvement percentage. In the case of the Customer database analysis, the estimated improvement is 63%. Clearly, a gain of 63% is a big jump in improvement, so the tool is doing something useful.

Based on the database and workload, the tool recommends that a clustered index be created for the State and Occupation columns and a nonclustered index for Occupation, FirstName, LastName, and EmailAddress. It also recommends that Statistics be created for State and Occupation as well as for FirstName.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 809

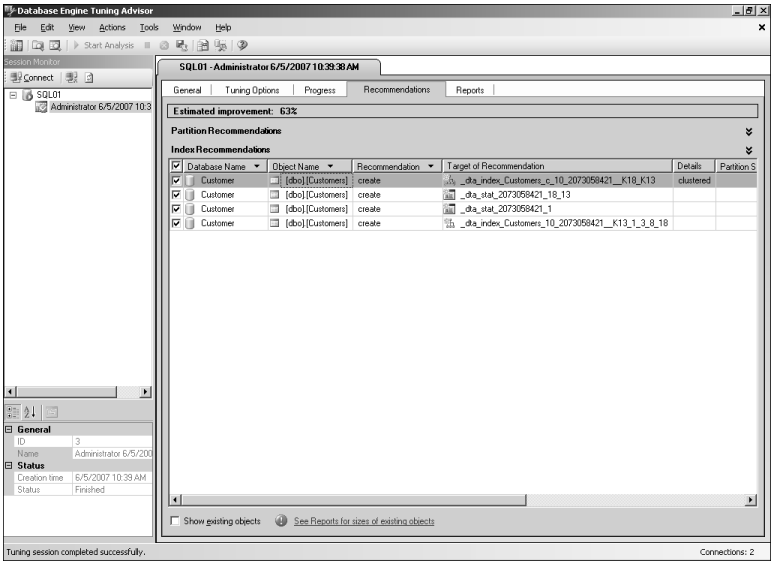

FIGURE 22.5
Database Engine Tuning Advisor recommendations.

For the recommended indexes, the tool shows the estimated size of the new indexes. This helps you plan for the additional space needed by the recommended indexes.

You can view the existing structures in the database along with the recommended objects by selecting the Show Existing Objects check box. This shows the existing four nonclustered indexes that already exist in the Customers database.

The last column in each recommendation, Definition, shows the definition for the object. These recommendation definitions are hyperlinks that show you the query needed to create the object. This information assists in the implementation of the recommendations.

Selecting the Reports tab shows the tuning summary and gives access to a collection of reports to interpret the recommendations. Table 22.10 shows the summary of the analysis.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 810

Table 22.10 **Database Engine Tuning Advisor Tuning Summary**

Description	Value
Date	6/5/2007
Time	10:51:48 AM
Server	SQL01
Database(s) to tune	[Customer]
Workload file	C:\data\Customer Database Trace.trc
Maximum tuning time	47 Minutes
Time taken for tuning	1 Minute
Expected percentage improvement	63.42
Maximum space for recommendation (MB)	3
Space used currently (MB)	2
Space used by recommendation (MB)	2
Number of events in workload	73
Number of events tuned	73
Number of statements tuned	4
Percent SELECT statements in the tuned set	100
Number of indexes recommended to be created	2
Number of statistics recommended to be created	2

In the Tuning Reports section, select the Statement Cost Report from the pull-down menu. The report shows the four query statements in the simulated workload. More importantly, it shows the percent improvement that the recommendations will have on each statement. For the more complex statement, the recommendations will generate an impressive 95.85% improvement in the performance.

Select the Workload Analysis Report from the pull-down menu. The report shows the number of query statements in the workload and the net impact of the tuning recommendation on the statements. In the case of the example, three statements would have a net decrease in cost and one would have no change in cost. Cost is measured in the time needed to execute the query. Depending on the recommendation, the cost might actually increase for some queries, as shown in the report.

Other reports show various aspects of the workload usage, such as which tables are used in the database and which columns in each table. These are useful for understanding how the data is being used by the workload.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 811

After reviewing the recommendations, you can apply the recommendations to the database.

Applying the Recommendations

The Database Engine Tuning Advisor tool provides several options for applying the recommendations:

- Cut/paste individual recommendations.
- Apply the recommendation from the tool.
- Save the recommendations to a file.

On the Recommendations tab, the Definition column of each recommendation is a hyperlink that pops up a window with the TSQL query needed to implement that specific recommendation. The window shows the specific code and has a Copy to Clipboard button to copy the code. This code can be pasted directly into the SQL Server Management Studio query window or any other TSQL query tool.

The easiest method of applying the recommendations is to select Actions, Apply Recommendations. This generates and runs the TSQL statements on the database to implement the recommended changes. They can be executed immediately or scheduled for a later date. Figure 22.6 shows the successful application of the recommendations to the Customer database.

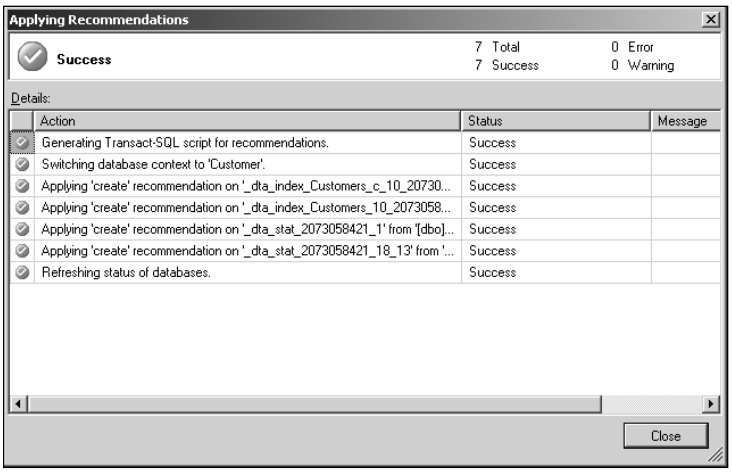

FIGURE 22.6
Applying the recommendations.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 812

The tool also allows the recommendations to be exported to a SQL file for execution at a later time or editing of the query statements. Select Actions, Save Recommendations to save the query statements to a file.

The saved recommendations for the Customer Database Tuning session are

```
use [Customer]
go
CREATE CLUSTERED INDEX [_dta_index_Customers_c_10_2073058421__
➥K18_K13]
       ON [dbo].[Customers]
       (
              [State] ASC,
              [Occupation] ASC
       )
       WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
➥DROP_EXISTING = OFF, ONLINE = OFF)
       ON [PRIMARY]
go

CREATE NONCLUSTERED INDEX
       [_dta_index_Customers_10_2073058421__K13_1_3_8_18]
➥ON [dbo].[Customers]
       (
              [Occupation] ASC
        )
        INCLUDE ( [FirstName],
        [LastName],
        [EmailAddress],
        [State]) WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY =
        OFF, DROP_EXISTING = OFF, ONLINE = OFF)
        ON [PRIMARY]
go

CREATE STATISTICS [_dta_stat_2073058421_1] ON
➥[dbo].[Customers]([FirstName])
go

CREATE STATISTICS [_dta_stat_2073058421_18_13] ON
➥[dbo].[Customers]([State], [Occupation])
go
```

These statements can be preserved for documentation or to apply to other databases in a replicated environment.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 813

Monitoring Processes with the Activity Monitor

The Activity Monitor in SQL Server Management Studio graphically displays information about

- Running processes
- Blocked processes
- Process and object locks
- User activity

Using this information, you can review activity on the SQL Server instance in real-time.

You can launch the Activity Monitor by following these steps:

1. Open SQL Server Management Studio and connect to a Database Engine.
2. Expand the Management folder and select the Activity Monitor.
3. Right-click on the Activity Monitor and select View Processes.

The Activity Monitor shows a list of the processes and information about each process. For performance troubleshooting, the three columns listed in Table 22.11 provide critical performance metrics.

Table 22.11 **Process Performance Metrics**

Column	Description
CPU	Cumulative CPU time for the process
Physical IO	Cumulative disk reads and writes for the process
Memory Usage	Number of pages in the procedure cache that are currently allocated to this process

Note

The CPU column is updated only when a query has been executed with `SET STATISTICS TIME ON`. When 0 is returned, `SET STATISTICS TIME` is `OFF`.

The view can be filtered by application, host, processes, status, database name, blocking, or if the process is waiting. This can narrow down the number of processes that you need to see in the Activity Monitor, for example, by filtering the view down to only processes that are blocked.

The status of the process is shown by the Activity Monitor. Table 22.12 shows the various status values possible in the Activity Monitor.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 814

Table 22.12 **Process Status Values**

Status (Icon)	Description
Running (Green Triangle)	The process is currently performing work.
Runnable (Green Check)	The process has a connection and has successfully run in the past. It currently has no work to perform.
Sleeping (Red Arrow)	The process has work to perform but is waiting for something, such as a lock or user input.
Background (Gray Arrows)	A background process that wakes up periodically to execute work.
Suspended (Hourglass)	The process has work to perform but has been stopped. The Status field does not contain the reason the process was suspended. The Wait Type field may contain information about why the process is suspended.
Other (Blue Dual Arrow)	The process is not in one of the other statuses.

Figure 22.7 shows the Activity Monitor for a busy application, in this case a Microsoft Operations Manager 2007 server. There are three runnable processes in the window, one suspended process, and the balance are sleeping processes.

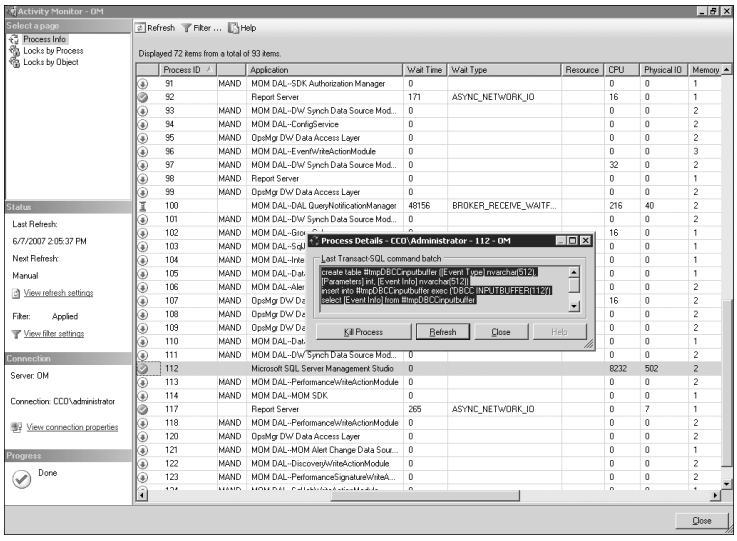

FIGURE 22.7
Activity Monitor.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 815

By right-clicking in the process, you can see the last TSQL command the process executed. Process 112 appears in the Process Details window in Figure 22.7. The process used 8232 CPU cycles and 502 disk reads/writes, and is currently using two pages in the memory cache. Compared to the other processes in the window, process 112 is using a lot of CPU and disk I/O.

Application Optimization and Troubleshooting

Application optimization and troubleshooting are typically beyond the scope of the database administrator. Application developers typically are responsible for the troubleshooting and optimization of applications.

However, you may find that certain query statements are consuming resources or taking a long time. This is typically discovered in database troubleshooting and optimization. Therefore, you can take key information and assist developers in their tasks.

Query Analysis with Execution Plan

The Query Editor in the SQL Server Management Studio allows you to analyze the execution plans of queries to determine their specific breakdown and costs for each step.

For example, consider the following query that runs against the Customer database:

```
USE Customer;
GO
SELECT FirstName, LastName, EmailAddress, Occupation,
➥YearlyIncome, City, State
        FROM dbo.Customers
        WHERE Gender = 'F' and MaritalStatus = 'S' and
        YearlyIncome = (SELECT MAX(YearlyIncome)
➥FROM dbo.Customers) ORDER BY City;
GO
```

This query essentially selects the highest-income single females in California and displays them by city. To analyze this query, follow these steps:

1. Launch the SQL Server Management Studio and open the Query Editor by selecting New Query.
2. Enter the preceding TSQL query into the editor.
3. Select Query, Display Estimated Execution Plan.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 816

Figure 22.8 shows the resulting graphical view of the query. The index scans are clearly the highest cost items at 45% each. The sort is the next highest at 7%. Optimizing this query could significantly reduce the cost.

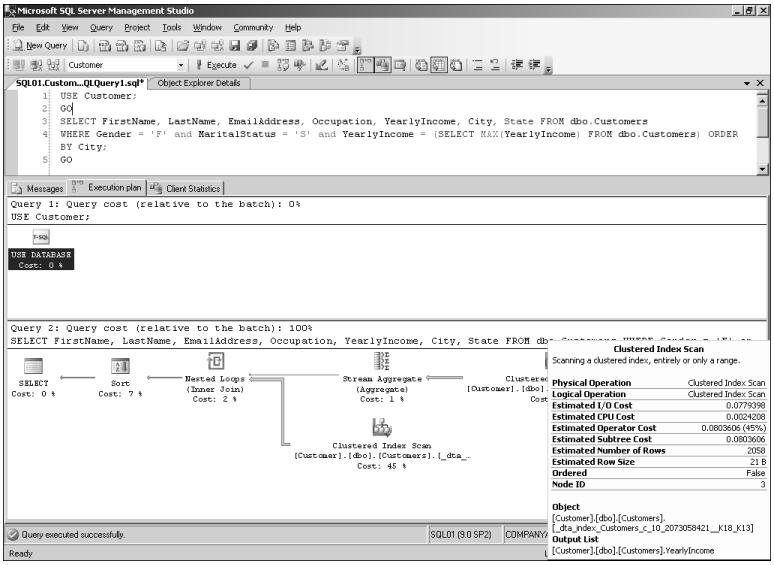

FIGURE 22.8
Estimated execution plan.

Following are some specific areas to look out for in the execution plan of a query:

- **Index or Table Scans**—They indicate that additional indexes are needed on the columns.
- **Sorts**—Sorting might be better done at the client or not at all.
- **Filters**—Filtering might require additional indexes, indicating that views are being used in TSQL or that there are functions in the statement.

Adjusting the query to smooth out the high-cost areas that the execution plan exposes can improve the performance of the application immensely.

If the database needs tuning, such as additional indexes, you can use the Database Engine Tuning Advisor to assist in that process.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 817

Query Analysis with the Tuning Advisor

Another method of troubleshooting a query is to use the Database Engine Tuning Advisor to analyze the query. To do this, follow these steps:

1. From the SQL Server Management Studio tool, select Query, Analyze Query in the Database Engine Tuning Advisor.
2. Everything is prepopulated by the SQL Server Management Tool, so select Actions, Start Analysis to generate recommendations.
3. Review the recommendations, which are to create a clustered index and to collect statistics.
4. Select Actions, Apply Recommendations to implement the recommendations.
5. Click Close to acknowledge the changes.
6. Close the Database Engine Tuning Advisor tool.

This procedure optimizes the database to run the query.

Back in the SQL Server Management Studio Query Editor, select Query, Display Estimated Execution Plan to see the difference in the query execution costs.

Cursors and the Lazy Developer

Another area to bring to the attention of application developers is the use of cursors. A `select` statement normally returns a set of rows called a *resultset*, and the application has to work with the entire resultset. A *cursor* is a database object that lets applications work with single rows within the resultset. The cursor allows

- Positioning within the resultset
- Retrieving a single row at the current position in the resultset
- Modifying the row at the cursor position in the resultset

Although this feature makes the developer's job easy, cursors are resource intensive. The use of cursors is normally associated with lazy development practices. Extensive use of cursors can affect the performance of the database, and their use is generally frowned on.

This information can all be passed on to the application developers for recommendations on how to optimize the application.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 818

Locking and Deadlock Detection

Locking allows users and processes to synchronize access to data and objects. This allows users in a multiaccess environment to prevent data from being corrupted. For example, when process A begins a transaction to modify a chunk of data X, the SQL Server Database Engine locks that data. If process B attempts to modify data X before the first transaction finishes, process B is paused (that is, sleeping). After process A finishes, process B is awakened and gets its lock on data X.

This process is all well and good, except when deadlocks occur. A deadlock condition occurs when two processes are mutually waiting on each other to free up locks on data. For example, consider if process A locks data Y and then requests a lock on data Z. At the same time, process B locks data Z and requests a lock on data Y. Both processes are paused and waiting for the other to free up the lock, which will never happen in theory and the processes will sleep forever; that is, the processes are deadlocked.

Resources that can deadlock include

- Locks
- Worker threads
- Memory
- Parallel queries
- Multiple active resultsets

Deadlocks can completely kill application performance and are unavoidable in a multiuser application such as SQL Server 2005.

Luckily, SQL Server 2005 has a mechanism for dealing with this condition. The SQL Server Database Engine runs deadlock detection to find and resolve these deadlock conditions. The deadlock detection is somewhat atavistic, basically selecting a deadlock victim and then resolving the situation by

- Terminating the ill-fated process's current batch command
- Rolling back the transaction, which frees all the locks it held
- Notifying the application with a 1205 error

The other, considerably luckier, process is then free to complete its transaction with the deadlock resolved. The deadlock victim is chosen on the basis of the least expensive transaction to roll back. This can be changed by setting the `DEADLOCK_PRIORITY`, which allows you or the application developer to force the selection.

The Database Engine executes deadlock detection on a periodic basis to locate deadlocks. The default interval is 5 seconds, but if a deadlock is

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 819

encountered, the process triggers more often until it stops detecting deadlocks. Then it reverts back to the 5-second interval.

Even though deadlocks are resolved, the delays that they cause affect performance. It is important to monitor SQL Server for errors and review the application code or logic if they occur frequently.

Summary

Many of the SQL Server 2005 problems that you are faced with are performance related. SQL Server 2005 is a complex application that is frequently charged with the handling of large datasets and real-time response requirements.

Fortunately, SQL Server 2005 contains the tools and instrumentation to allow you to easily conduct performance troubleshooting. They include the SQL Server Profiler and the Database Engine Tuning Advisor, which inject artificial intelligence into the performance-tuning process, providing both guidance and implementation features.

Best Practices

Some important best practices from the chapter include

- Conduct performance optimization and troubleshooting at least once a year.
- Use the SQL Server Profiler to gather workloads over at least a 24-hour period.
- Use the Database Engine Tuning Advisor to recommend indexes, statistics, and partitions based on the workloads.
- Apply service packs in a timely manner to take advantage of performance improvements.
- Specifically, apply Service Pack 2 as soon as possible.
- Use clustered indexes.
- Don't use cursors if at all possible.
- Use Operations Manager 2007 to automate the collection of performance metrics and to maintain a historical record of the data.
- Use 64-bit SQL Server 2005 for memory-intensive applications such as Analysis Services.
- Configure logging on packages and data flow tasks in Integration Services to monitor performance of packages.

29_0672329565_ch22.qxd 9/7/07 8:40 AM Page 820