HOW TO: Troubleshoot Application Performance with SQL Server

Article ID: 224587 - View products that this article applies to.

This article was previously published under Q224587

SUMMARY

This step-by-step article describes how to troubleshoot SQL Server performance issues. Troubleshooting performance issues involves the use of a series of steps to isolate and determine the cause of an application slowdown. Possible causes include:

- Blocking.
- System resource contention.
- Application design problems.
- A particular set of queries or stored procedures with long execution times.

This article describes how to determine the source of a performance issue. It also references other articles in the Microsoft Knowledge Base that cover the details of specific performance issues for additional troubleshooting.

SQL Profiler

SQL Profiler is a powerful tool when troubleshooting your SQL Server 7.0, or later, application performance. SQL Profiler allows you to easily capture all the events that are occurring on the server under a typical load, and it provides information about them. Using SQL Profiler in conjunction with Microsoft Windows NT Performance Monitor and some simple queries to identify whether blocking is occurring will give you the information you must have to resolve the vast majority of performance problems.

What to Monitor

1. Set up SQL Profiler to capture a trace. To do so, follow these steps:

a. Open SQL Profiler.
b. On the **Tools** menu, click **Options**.
c. Make sure that the **All Event Classes** and the **All Data Columns** options are selected.
d. Click **OK**.
e. Create a new trace.
f. On the **File** menu, point to **New**, and then click **Trace**.
g. On the **General** tab, specify a trace name and a file to capture the data to.
h. On the **Events** tab, add the following event types to your trace:

Heading	Event to add	Description
Cursors	CursorPrepare	This event indicates that a cursor on an SQL statement has been prepared by using ODBC, OLEDB, or the DB-Library.
Error and Warning	Missing Column Statistics	This event indicates column statistics that might have been useful for the Optimizer were not available. The **Text** column shows the list of columns with missing statistics. This event, in conjunction with a **Misc: Auto-UpdateStats** event, indicates that the **Auto Create Statistics** option was triggered.
Misc.	Attention	This event indicates that an attention signal was sent by a client.
Misc.	Auto-UpdateStats	This event indicates that the **Auto Update Statistics** option was triggered.
Misc.	Exec Prepared SQL	This event indicates that ODBC, OLE DB, or the DB-Library executed a previously prepared Transact-SQL statement, or statements.
Misc.	Execution Plan	This event shows the plan tree of the Transact-SQL statement that was executed.
Misc.	Prepare SQL	This event indicates that an ODBC, OLE DB, or DB-Library application prepared a Transact-SQL statement, or statements, for use.
Misc.	Unprepare SQL	This event indicates that an ODBC, OLE DB, or DB-Library application unprepared a Transact-SQL statement, or statements, for use.
Sessions	Connect	This event indicates that a new connection has been made.
Sessions	Disconnect	This event indicates that a client has disconnected.
Sessions	Existing Connection	This event indicates that a connection existed when the SQL Profiler trace was started.
Stored Procedures	SP: Completed	This event indicates when a stored procedure has completed execution.

Stored Procedures	SP: Recompile	This event indicates that a stored procedure was recompiled during execution.
Stored Procedures	SP: Starting	This event indicates when a stored procedure has started execution.
Stored Procedures	SP: StmtCompleted	This event indicates when a statement in a stored procedure has completed execution.
TSQL:	SQL:BatchCompleted	This event indicates that a Transact-SQL batch completed. The **Text** column shows the statement that was executed.
TSQL:	SQL:StmtCompleted	This event indicates a Transact-SQL statement completed. The **Text** column shows the statement that was executed.
TSQL:	RPC:Completed	This event indicates that a remote procedure call (RPC) has completed.

i. If your application is receiving timeout errors, stops responding (hangs), or experiences other events that cause the problem statements to never complete, also include the following events:

TSQL:	SQL:BatchStarting	This event indicates the start of a Transact-SQL batch. The **Text** column shows the statement being executed.
TSQL:	SQL:StmtStarting	This event indicates the start of a Transact-SQL statement. The **Text** column shows the statement being executed.
TSQL:	RPC:Starting	This event indicates the start of a remote procedure call (RPC).
Stored Procedures	SP: StmtStarting	This event indicates when a statement in a stored procedure is starting execution.

This will help to make certain that you can see the statement that was executing when the timeout occurred

j. On the **Data Columns** tab, make sure that the following columns are included:

For SQL Server 2000

Start Time

End Time

LoginSid

SPID

Event Class

TextData

IntegerData

BinaryData

Duration

CPU

Reads

Writes

Application Name

NT User Name

DBUserName

For SQL Server 7.0

Start Time

End Time

Connection ID

SPID

Event Class

Text

Integer Data

Binary Data

Duration

CPU

Reads

Writes

Application Name

NT User Name

SQL User Name

For information about using SQL Profiler, see SQL Server 7.0 and SQL Server 2000 Books Online.

2. Use Performance Monitor to capture Windows NT and SQL Server counters. To do so, follow these steps:

a. Start Windows NT Performance Monitor.
b. On the **View** menu, click **Log**.
c. On the **Options** menu, click **Log**.
d. Specify a file name and location to log the performance counters. You can adjust the update interval as appropriate.
e. On the **Edit** menu, click **Add To Log**.
f. Add all objects. Both the Windows NT and the SQL Server objects.
g. To start the log, on the **Options** menu, click **Log**, and then click the **Start Log** button.

For additional information, click the following article number to view the article in the Microsoft Knowledge Base:
150934 (http://support.microsoft.com/kb/150934/EN-US/) How to Create a Performance Monitor Log for NT Troubleshooting

3. Check for blocking.

To see if blocking is occurring, run the **sp_who** system stored procedure:

```
exec sp_who
```

This output will contain a **blk** column. Examine the output for any non-zero entries that indicate that blocking is occurring. Run this procedure periodically throughout the timeframe when the performance slowdown is occurring.

Note Running the **sp_who** system stored procedure is a check to see if blocking exists. Typically, it is not enough information to fully troubleshoot a blocking problem. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:
251004 (http://support.microsoft.com/kb/251004/EN-US/) INF: How to Monitor SQL Server 7.0 Blocking

Run the Application Under Typical Load

Ideally, it is best to capture the SQL Profiler, Performance Monitor, and blocking output during the same timeframe. This timeframe must encompass a time when application performance goes from good to bad. The combination of this information will help you to get a clearer picture of where the performance slowdown is occurring.

Interpret the Results

1. Check for blocking.

   If the **blk** column in the **sp_who** output is non-zero, this indicates that blocking is occurring on your system. If processes are blocking each other, the processes that are being blocked can experience longer execution times. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:
   224453 (http://support.microsoft.com/kb/224453/EN-US/) INF: Understanding and Resolving SQL Server 7.0 or 2000 Blocking Problems

2. Examine the SQL Profiler output.

   Viewing SQL Profiler data efficiently is extremely valuable in resolving performance issues. The most important thing to realize is that you do not have to look at everything you captured. Be selective. SQL Profiler provides capabilities to help you effectively view the captured data. On the **Properties** tabs (click **Properties** on the **File** menu), SQL Profiler allows you to limit the data displayed by removing data columns or events, grouping (sorting) by data columns, and applying filters. You can search the whole trace or only a specific column for specific values (on the **Edit** menu, click **Find**). You can also save the SQL Profiler data to a SQL Server table (on the **File** menu, point to **Save As**, click **Trace Table**), and then run SQL queries against it.

   Be careful that you perform filtering only on a previously saved trace file. If you perform these steps on an active trace, you risk losing data that has been

captured since the trace was started. Save an active trace to a file or table first (on the **File** menu, click **Save As**) and then reopen it (on the **File** menu, click **Open**) before you continue. When you work on a saved trace file, the filtering does not permanently remove the data that is being filtered out; it just does not display it. You can add and remove events and data columns as needed to help focus your searches.

The first step in examining SQL Profiler trace files for performance cases is to determine where the different types of events are occurring on the server.

Group the trace by Event Class:

a. On the **File** menu, click **Properties**.

b. On the **Data Columns** tab, use the UP button to move **Event Class** under the **Groups** heading, and the DOWN button to remove all other columns under the **Groups** heading.

c. Click **OK**.

Grouping by the event class column shows what type of events are occurring on SQL Server, and how frequently. Search this column for the following events:

SP:RECOMPILE

This event indicates that a stored procedure was recompiled during execution. Several recompile events indicate that SQL Server is spending resources on query compilation instead of query execution.

For additional informationabout troubleshooting stored procedure recompilations, click the following article number to view the article in the Microsoft Knowledge Base:

243586 (http://support.microsoft.com/kb/243586/EN-US/) INF: Troubleshooting Stored Procedure Recompilation

Attention

An attention signal indicates that a query was canceled by a client. This is generally because of one of two causes:

The user explicitly canceled the query or ended the application.

-or-

A query timeout was exceeded.

If you see attention signals, this may indicate that certain queries are running slowly.

For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

243589 (http://support.microsoft.com/kb/243589/EN-US/) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

To help identify the query that received the attention signal, revise the trace so that it is not grouped by any data column, and filter on the system process ID (SPID) that received it (on the **Filters** tab, set SPID = *x*). The **SQL:StmtStarting**, **SQL:BatchStarting**, or **SP:StmtStarting** event immediately preceding the attention signal is the query that received the timeout or cancel. You can search the **Event Class** column for the Attention event to easily locate it (on the **Edit** menu, click **Find**).

PREPARE SQL and EXEC PREPARED SQL

The **Prepare SQL** event indicates that an ODBC, OLE DB, or DB-Library application prepared a Transact-SQL statement ,or statements, for use. The **Exec Prepared SQL** event indicates that the application made use of an existing prepared statement to run a command.

Compare the number of times these two events occur. Ideally, an application must prepare a SQL statement one time and run it several times. This saves the Optimizer the cost of compiling a new plan each time the statement is executed. Therefore, the number of **Exec Prepared SQL** events should be much larger than the number of **Prepare SQL** events. If the number of **Prepare SQL** events is roughly equivalent to the number of **Exec Prepared SQL**events, this may indicate that the application is not making good use of the prepare/execute model. It is best not to prepare a statement that is only going to be executed a single time. For more information about preparing SQL statements, see the "Preparing SQL Statements" topic in SQL Server 7.0 Books Online.

If the number of **Exec Prepared SQL** events is not three to five times greater than the number of **Prepare SQL** events, the application may not be making efficient use of the prepare/execute model. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

243588 (http://support.microsoft.com/kb/243588/EN-US/) HOW TO: Troubleshoot the Performance of Ad-Hoc Queries

In SQL Server 2000, the excessive roundtrips per prepare/execute will be eliminated, so the 3-5 ratio is not as stringent. However, it can still be a good rule, to try and reuse the prepared plan more than one time.

Missing Column Statistics

This event indicates that statistical information the Optimizer could have used to generate a better query plan was unavailable. This indicates that the query does not have useful indexes on at least one table involved. Beyond not having a useful index, SQL Server does not even have statistical data about the columns involved to make an informed decision for a query plan. The outcome is that the query plan generated may not be the optimal one. If you see these events, look at the query and the execution plan generated, and then see the following article in the Microsoft Knowledge Base for steps to take to improve the performance of this query:

243589 (http://support.microsoft.com/kb/243589/) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

When you view the **Missing Column Statistics** events, focus first on those that occur in association with long-running queries. Some events may be generated and resolved automatically by SQL Server with autostats and may not require user intervention. Therefore, the best strategy is to first focus on queries with long duration, as shown later in this article, and note if there are associated **Missing Column Statistics** events.

If you are not seeing instances of these event classes, the next step is to determine where the time is being spent.

Group the trace output by Duration:

a. On the **File** menu, click **Properties**.

On the **Data Columns** tab, use the UP button to move **Duration** under the **Groups** heading, and the DOWN button to remove all other columns under the **Groups**heading.

c. On the **Events** tab, remove all groups except **TSQL** and **Stored Procedures**.

d. Click **OK**.

By grouping on duration, you can easily see what SQL statements, batches, or procedures are running the slowest. It is very important to look not only at the time when the problem is occurring, but also to get a baseline of when performance is good to compare against. You can filter on start time to break the trace up into sections when performance was good, and a separate section for when performance was poor. Look for the queries with the longest duration when performance is good. These are most likely the root of the problem. If overall system performance is decreased, even good queries can show long durations as they are waiting on system resources.

If you see a small number of queries with high durations, see the following article in the Microsoft Knowledge Base:

243589 (http://support.microsoft.com/kb/243589/) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

If you see that the duration of individual queries is low, but there are several of them, and the **SQL Compilations/sec** counter in Performance Monitor output (described later) is high, see the following article in the Microsoft Knowledge Base:

243588 (http://support.microsoft.com/kb/243588/) HOW TO: Troubleshoot the Performance of Ad-Hoc Queries

Examine the remaining data columns:

Additional insight into the nature of a performance problem can be gained by viewing the other data columns in the trace data. Here are some things to consider:

If the CPU usage is high, group by CPU to see what queries are the biggest users of CPU time. Search the **Text** column for "hash" or "merge" to find what query execution plan is using these join types. They are more CPU and memory intensive than a nested loop join, which is generally IO intensive.

If disk IO is the bottleneck, group by reads and writes. View the **Application Name**, **NT User Name**, and **SQL User Name** fields to help isolate the source of a long-running query.

The integer data column of the exception event will indicate any errors that were returned back to the client. You can find the text of the error message by searching on the number in SQL Server 7.0 Books Online.

The **Connection ID** field is helpful to make sure that you are looking at the same sessions for a specific client. A SPID cannot guarantee this, as a user may have disconnected and a new user connected and received the same SPID.

The benefit derived from these fields may vary depending on the scenario, but they should be examined if the obvious fields earlier in this article do not provide an answer.

3. Examine the Performance Monitor output.

Performance Monitor will show you the overall system bottlenecks. It may be that SQL Server and the application are performing as expected, but the computer is underpowered, lacking memory or other resources. Or certain counters can indicate problems with the way the application and SQL Server are performing. At a minimum, check the following counters:

- Object: Process

  Counter: Processor

Instance: SQL Server

- Object: Processor

  Counter: %Processor Time

  Instance: Check each processor instance

- Object: Physical Disk

  Counter: Avg. Disk Queue Length

  Instance: Check each physical disk instance

- Object: SQL Server:SQL Statistics

  Counter: SQL Compilations/sec

Look for a trend over the timeframe from when performance went from good to bad: what increased first? Is the computer CPU bound or DISK IO bound? This information, together with the Profiler output earlier in this article, will help you narrow down the problem areas. High CPU problems may indicate large numbers of stored procedure recompilations, ad-hoc query compilations, or intensive use of hash and merge joins. The articles referenced earlier in this article must be followed to determine the correct course of action. High disk queue lengths may indicate the need for more system memory or an improved disk subsystem.

Properties

Article ID: 224587 - Last Review: October 26, 2007 - Revision: 4.3

APPLIES TO

- Microsoft SQL Server 7.0 Standard Edition

Keywords: kbproductlink kbhowtomaster kbhowto kbinfo KB224587