l e x a n a \ n e t

September 2008

Using SQL Server’s Profiler to Help Understand Legacy Software

By Drew Hamre

email

Profiler is one of the many tools bundled with SQL Server, Microsoft’s flagship database management system. Profiler offers a friendly graphical interface for the monitoring and trace facilities of the DBMS. You can use Profiler to capture data about server activity, and later analyze this information to isolate deadlocks, identify poorly performing stored procedures, capture long-running queries, collect data for index tuning, and so on.

Although Profiler is most commonly used to diagnose performance problems, the detailed information it provides has other uses. In a recent engagement, Profiler proved indispensable in helping understand the behavior of legacy COBOL software that was being migrated during a re-hosting project. This paper will review our experiences using Profiler and the benefits it provided during this transition.

The Dark Mysteries of 25-Year-Old Code

The re-hosting project was motivated by the approaching end-of-life of the client’s core infrastructure, which ran on UNIX hardware and used TurboIMAGE, a high performance network (CODASYL) database manager. The project’s goal was to move all active applications from this legacy environment to new Wintel hardware running Windows Server 2003 and SQL Server 2005.

The client’s legacy applications were written in COBOL (the familiar old workhorse, though extended with operands for the PROTOS code generation product). By contractual agreement, applications were ported with only the minimal conversion necessary to run in the new environment. Thus, COBOL software would remain written in COBOL (rather than being re-written in, say, C#). Within this restriction, COBOL code was ported to a Windows-friendly dialect of the language (AcuCobol).

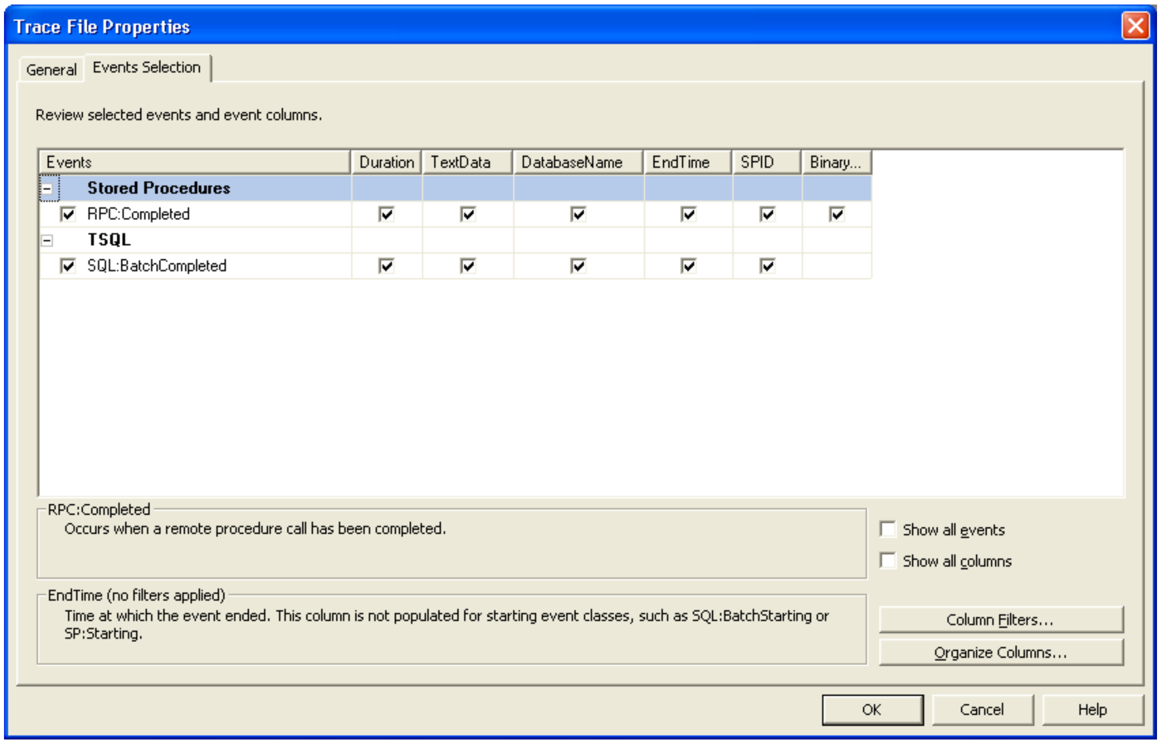
Consultants developed a sophisticated data dictionary system to translate the TurboIMAGE file definitions/datatypes into correlated SQL Server schema and DDL scripts. The system included a pre-compiler that referenced this data dictionary while scanning PROTOS/COBOL code, translating TurboIMAGE read/write calls into corresponding sets of embedded SQL statements.

The client’s software was formidable not only due to the unfamiliar (to many of us) technology, but also due to the code’s sheer volume and age. One telling statistic: the pre-compiler needed to generate more than 20,000 different COBOL include files (more than 500-Megabytes of text) to fully capture the functionality of the original system. Much of the legacy software was several decades old, and the software's creators had long since left the client's employ. If the re-hosting team had questions about the code's inner workings, answers would need to come from elsewhere.

Using SQL Profiler

By default, Profiler is launched from *Start* | *Programs* | *Microsoft SQL Server 2005* | *Performance Tools* | *SQL Server Profiler*. The user then a) directs Profiler to connect to a SQL Server instance and b) provides parameters for a new trace session. These parameters tell Profiler which server events and associated data elements to record.

To simplify the event/data selection process, Profiler comes pre-packaged with a collection of templates that support common measurement scenarios. During the re-hosting project, our profiler usage was often built upon one of the simplest templates -- *TSQL_Duration*. This template includes metrics that capture the text of SQL statements that are executed (including stored procedures) and the execution time for each. This template is intended chiefly for performance optimization.



Review selected events and event columns.

Events	Duration	TextData	DatabaseName	EndTime	SPID	Binary...
Stored Procedures						
✓ RPC:Completed	✓	✓	✓	✓	✓	✓
TSQL						
✓ SQL:BatchCompleted	✓	✓	✓	✓	✓	

RPC:Completed
Occurs when a remote procedure call has been completed.

EndTime (no filters applied)
Time at which the event ended. This column is not populated for starting event classes, such as SQL:BatchStarting or SP:Starting.

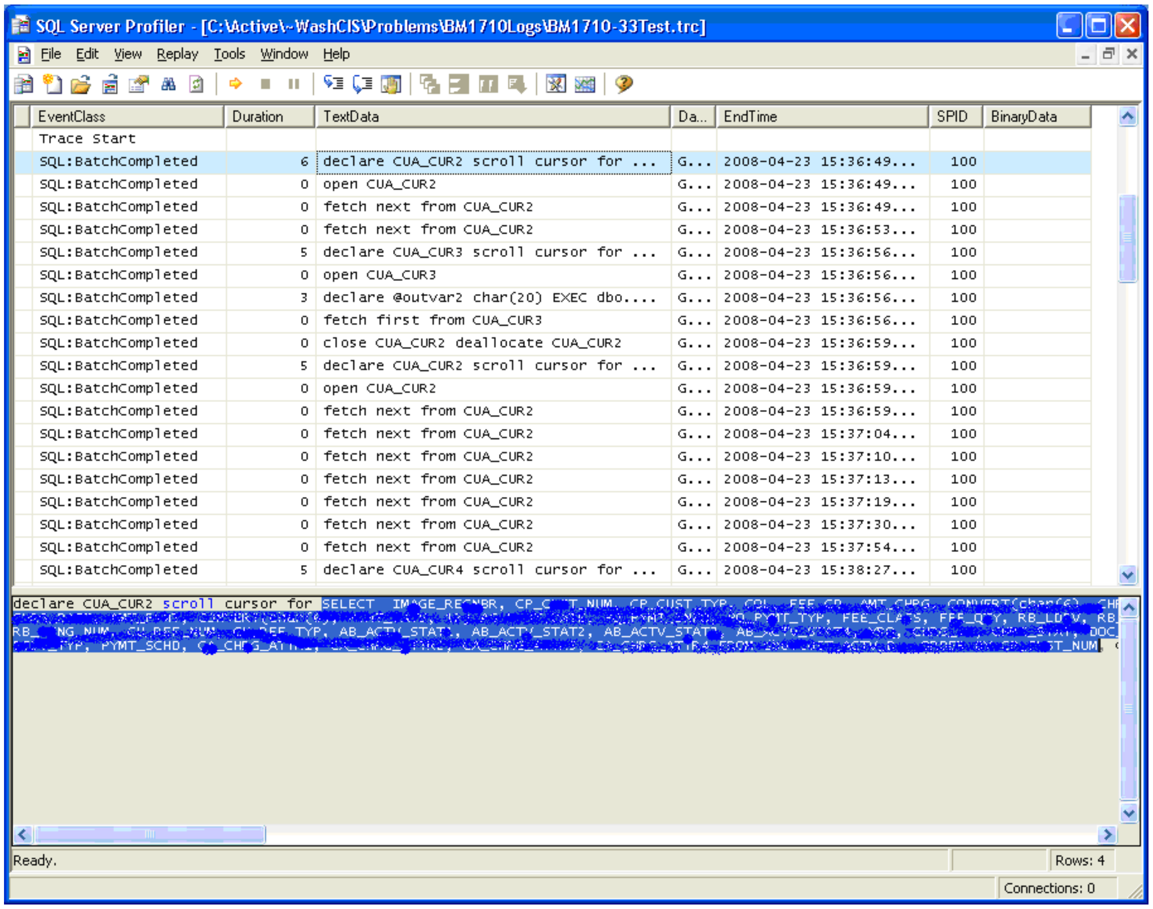
Figure 1: Above is the re-hosting project's most commonly used Profiler template, which builds upon the pre-defined *TSQL_Duration* template

This template's simplicity reflects the single most important principle underlying recommendations for using Profiler effectively -- ***Think Small***:

- Minimize the amount of information collected by limiting the number of events and data columns recorded.
- Limit the amount of time Profiler actually runs; for example, during the re-hosting project we would start/stop Profiler in concert with COBOL debugger breakpoints, activating Profiler for recognized code 'hot spots'.

- Limit Profiler's impact on shared system resources by running it remotely (that is, launch Profiler from a computer that has access to, but is not hosting, the target DBMS instance).
- Minimize Profiler's own demands on the database engine by writing traces to the client file system, rather than writing traces to a database table.
- Actively filter the trace during recording so that only target database and/or process activity is recorded, rather than recording activity for all databases/processes and filtering at a later time.

Using the template in Figure 1, information similar to the following is captured:


SQL Server Profiler - [C:\Active\~WashCIS\Problems\BM1710Logs\BM1710-33Test.trc]
File Edit View Replay Tools Window Help

EventClass	Duration	TextData	Da...	EndTime	SPID	BinaryData
Trace Start						
SQL:BatchCompleted	6	declare CUA_CUR2 scroll cursor for ...	G...	2008-04-23 15:36:49...	100	
SQL:BatchCompleted	0	open CUA_CUR2	G...	2008-04-23 15:36:49...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:36:49...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:36:53...	100	
SQL:BatchCompleted	5	declare CUA_CUR3 scroll cursor for ...	G...	2008-04-23 15:36:56...	100	
SQL:BatchCompleted	0	open CUA_CUR3	G...	2008-04-23 15:36:56...	100	
SQL:BatchCompleted	3	declare @outvar2 char(20) EXEC dbo....	G...	2008-04-23 15:36:56...	100	
SQL:BatchCompleted	0	fetch first from CUA_CUR3	G...	2008-04-23 15:36:56...	100	
SQL:BatchCompleted	0	close CUA_CUR2 deallocate CUA_CUR2	G...	2008-04-23 15:36:59...	100	
SQL:BatchCompleted	5	declare CUA_CUR2 scroll cursor for ...	G...	2008-04-23 15:36:59...	100	
SQL:BatchCompleted	0	open CUA_CUR2	G...	2008-04-23 15:36:59...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:36:59...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:04...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:10...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:13...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:19...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:30...	100	
SQL:BatchCompleted	0	fetch next from CUA_CUR2	G...	2008-04-23 15:37:54...	100	
SQL:BatchCompleted	5	declare CUA_CUR4 scroll cursor for ...	G...	2008-04-23 15:38:27...	100	

```
declare CUA_CUR2 scroll cursor for SELECT [redacted]
```

```
Ready.                                   Rows: 4
                                         Connections: 0
```

Figure 2: Example of Profiler results tracing SQL statement execution.

In this example, SQL statements are captured and displayed serially in order of execution. Because data for the re-hosted applications resides almost entirely in SQL Server (INI files were a common exception), the trace (which captures each SQL statement issued by the program) gives a comprehensive picture of the software's I/O activity.

With this information in hand, analysts can manually re-create and examine the recordsets that are processed by the program. All they need do is select and highlight SQL statements of interest (Figure 2), and copy/paste the relevant SQL text from Profiler into an interactive query tool.

Manipulating Profiler output: Re-loading into SQL Server

Even if Profiler output is constrained (recommendations above), traces may still be too massive to be comfortably analyzed. Summarizing traces with an automated editing tool can make the information more intelligible. Third-party products like UltraEdit or WinGREP are useful here, and SQL Server can also fill this role.

To use SQL Server, simply double-click the earlier-created trace (*.trc) file to launch Profiler. Next, choose the menu option *File* | *Save As …* | *Trace Table …* to save the trace in a database table freshly created for this purpose.

Once loaded, it's easy to add additional summary columns and to make *ad hoc* edits that (for example) isolate all DECLARE CURSOR statements, highlight the names of tables referenced, summarize recurring SELECT statement patterns, and so on.

```
-- Example: Populate new summary column 'cursorname'
set cursorname = substring(textdata,9,20)
where textdata like 'DECLARE%'

-- Example: Recode SELECTs based on table/restriction references
update worktable
set eventclass = 11
where fromclause like '%datatable01%'
and whereclause like '%cust_number01%'
```

Figure 3: Examples of *ad hoc* edits summarizing Profiler data

In this way, the analyst can construct a fairly readable view of the program's I/O behavior – the tables and keys it accessed, its selection constraints, and its joins.

Using SQL Profiler to Understand Legacy Software

Profiler was a popular diagnostic tool throughout the re-hosting project, especially during unit/integration testing. Our focus was the client's batch sub-system, comprised chiefly of long-running report/analytic applications. Performance tended to be I/O-bound, as is typical for such programs. Note that AcuCobol requires *cursors* be used to move data between SQL Server and COBOL.

Despite the cursor requirement (an efficiency risk), runtime performance of the re-hosted system was generally excellent. A handful of slow-performing exceptions benefitted from Profiler analysis, as discussed below.

Which table drives looping behavior, and how many rows are processed? -- Reporting applications are typically driven by entities from a single table. A common implementation pattern would be an outer loop that reads a row from (say) a customer table, then retrieves related information from supplemental tables, and then reads the next customer and so on until all active customers are processed.

This central table (and its active subset of records) often isn't apparent from the source code. Luckily, Profiler traces provided this information. Analysts could take the captured SQL statements and re-create the exact rows processed by the program. Once identified, analysts could use the information to build test 'subset' databases. In a personal development copy of the database, an analyst would edit this master table so it contained far fewer rows (Figure 4). This allowed processing in minutes, rather than hours, speeding further analysis.

```
-- Initially how many records?
select count(*) from datatable

-- Delete a RANDOM subset of 250000 records from datatable
drop table #TEMPTABLE
GO

select top 250000 primary_key_name into #TEMPTABLE
from datatable
order by newid()
go

delete from datatable
where primary_key_name in (select primary_key_name from #TEMPTABLE)
go

drop table #TEMPTABLE
GO

-- How many records remain?
select count(*) from datatable
go
```

Figure 4: Sample SQL Server code to delete random subset of records from a table.

Cache management: Are we re-reading the same record? -- One of the reasons the SQL Server emulation of TurboIMAGE performed so well was due to the emulation library's sophisticated cache management. However, in one case an uncommon retrieval sequence caused the cache to be reloaded repeatedly. Profiler traces showed the software rebuilding and serially reading the same cursor set (for example, rebuilding the cursor set and issuing five 'FETCH NEXTs' for the fifth record; then rebuilding the cursor set and issuing six 'FETCH NEXTs' for the sixth record, and so on in a quasi-factorial progression). Once Profiler helped identify the problem, the revised program ran faster by several orders of magnitude.

Implicit sorts when emulating CODSYL -- Profiler uncovered another more subtle emulation issue when traces revealed the COBOL software was expecting a secondary implicit sort order (an undocumented feature of the legacy software navigation through its network database). This implicit sort was emulated by adding a secondary sort key to re-hosted software, fixing the problem.

Reviewing data issues -- Among the data conversion issues between TurboIMAGE and SQL Server, none caused such consistent headaches as DATETIME data, a problem exacerbated by the age of the legacy software (which pre-dated Y2K awareness, and which included layers of subsequent patches). In addition, coding for unavailable or inapplicable dates was inconsistent.

Profiler proved indispensable for reviewing data issues. The tool allowed analysts to capture the offending SQL statement, re-create the result set, and then simply skim the returned fields looking for outliers.

Avoiding the Gigabyte Trace File

Despite its utility, Profiler should be used carefully. Traces are multi-threaded and can consume the lion's share of cycles on even quad-CPU monitoring systems. If you're running Profiler on a shared system (e.g., from within a Terminal Server session), the gesture of dropping the Profiler process priority would be appreciated by your fellow users. As noted above, the great speed and many options of Profiler make it easy to create massive trace files that are too unwieldy to be useful. The engagement's 'high-water-mark' was a gigabyte trace that sat unused for months until finally being deleted.

The goal in using Profiler should be to create small, readable traces that make the software's 'narrative' apparent. If you succeed and create traces that are readable, you'll find Profiler has a story to tell.

Acknowledgement

Thanks to reviewers for their time and suggestions; any inaccuracies remain mine.

References and Other Resources

SQL Server 2005 Books Online: SQL Server Profiler Reference
http://msdn.microsoft.com/en-us/library/ms173757.aspx

Advanced Tips for SQL Server 2005 Profiler & Dashboard by Brad M. McGehee
http://www.duringlunch.com/file.axd?file=Riverside+Profiler+Dashboard+Presentation.ppt

SQL Server Query Performance Tuning Distilled, Second Edition by Sajal Dam (2004). APress. ISBN (pbk):1-59059-421-5

Summary

SQL Server Profiler is commonly used to optimize the performance of database software. However, Profiler's detailed traces support other needs, and the tool may be helpful in gaining an understanding of undocumented legacy software. This paper presents tips for using Profiler efficiently, and discusses Profiler's benefits in helping understand decades-old COBOL software during a recent re-hosting project.

Drew Hamre is a principal of lexana\net and lives in Golden Valley, Minnesota.