SQL Server: Introduction to Extended Events
Module 9: Basic Troubleshooting Scenarios

Jonathan M. Kehayias
Jonathan@SQLskills.com

Introduction

- **Like SQL Trace, Extended Events should be used in situations where they provide the best method of identifying problems**

- **Some basic troubleshooting scenarios where Extended Events can be used include:**
  - Identifying deadlock issues
  - Identifying blocking issues
  - Identifying recompilation issues
  - Identifying when errors occur
  - Tracking session-level waits

Identifying Deadlock Issues

- **The xml_deadlock_report event fires when the Lock Monitor in SQL Server identifies a deadlock and raises error 1205 (when killing the deadlock victim process/processes)**
- **XML report that contains all of the information needed to troubleshoot deadlocks**
- **New XML format in Extended Events**
  - Supports multi-victim deadlock analysis
  - Incompatible with graphical display of deadlock graph in SSMS
  - Reduces redundant information that existed in previous XML format
- **The RTM releases of SQL Server 2008 and 2008R2 contain a bug which causes the new XML format to be incorrectly formed**
  - SQL Server 2008 SP1+CU6 or higher or SQL Server 2008R2 RTM+CU1 or higher fix this bug (http://support.microsoft.com/kb/978629)
- **Collected by default on all instances of SQL Server 2008 onwards in the system_health session**

Identifying Blocking Issues

- **The blocked_process_report event fires based on the value configured for the 'blocked process threshold' sp_configure option in the SQL Server**
- **XML report that contains information about the blocking and blocked processes in a blocking scenario for further debugging to identify and prevent the problem**
- **Setting the 'blocked process threshold' too low can result in excessive event generation**
  - For example, if the threshold is set at 10 seconds and a blocking scenario lasts for 38 seconds, three blocked_process_report events will be generated (one every 10 seconds)
  - In the same example, if there are multiple blocked sessions in a blocking chain, each blocked session will generate a blocked_process_report event every 10 seconds

Identifying Recompilation Issues

- **The sql_statement_starting and sp_statement_starting events contain a ‘state’ column that specifies whether the statement was recompiled during execution**
    - The state column is a mapped to the statement_starting_state map and provides three values: Normal, Recompiled, and Execution Plan Flush
    - Recompilation causes the event to fire twice: once for state=Recompiled and once for state=Normal
- **The sql_statement_recompile event fires for any statement-level recompilation in the system**
    - Ad hoc batches, stored procedures, and triggers are included
    - The recompile_cause column is mapped to the statement_recompile_cause map and provides the reason the recompile occurred

Identifying When Errors Occur

- **The error_reported event fires when errors occur during execution, even if the error is handled by Transact-SQL code**
  - The is_intercepted column can be used to determine if the error was handled by a TRY/CATCH block in the Transact-SQL code
- **Using filters on the error_number, severity, and/or state can make it easier to identify the root cause of specific problems**
  - 220 - Arithmetic overflow error for data type…
  - 8152 - String or binary data would be truncated…
  - 2627 - Cannot insert duplicate key in object…
- **Actions like the tsql_stack, query_hash, and sql_text can make identifying the point in code where the error occurred possible**
  - Prevents collecting sql_statement_starting and/or sp_statement_starting events to correlate where the error is occurring

Tracking Session Wait Statistics

- Understanding the causes of waits inside SQL Server can help identify performance bottlenecks and potential future problems
- The wait_info and wait_info_external events fire whenever a task has to wait during its execution
- Predicates on the session_id global field can allow tracking waits for a specific session in the server, or can be used to sample all sessions on the server
- An example of how to track session level waits is in Module 3 of the SQL Server: Performance Troubleshooting Using Wait Statistics course

Viewing Historical System Health

- **In SQL Server 2012, sp_server_diagnostics executes in five minute intervals to check the current health of the SQL Server instance**
- **It provides four sections of health information:**
  - System – spinlock backoffs, CPU usage, memory dumps, non-yielding tasks
  - Resource – process and memory manager memory usage
  - Query Processor – max workers, current number of workers, idle workers, tasks completed in interval, top 10 non-preemptive and preemptive wait types by count
  - I/O subsystem – latch timeouts, long interval I/O, pending requests
- **The output of sp_server_diagnostics is collected by the system_health session and written to a file in the file system**
  - Data can be trended over longer periods of time
  - Data will be available if a crash or restart occurs for the instance

Summary

- **Basic troubleshooting can be accomplished using Extended Events**
- **In some cases the data necessary for troubleshooting problems, like deadlocks occurring, is already being collected by the system_health event session on the server**

- **Many more scenarios for using Extended Events exist than were covered in this module**
  - These are merely suggested scenarios where basic troubleshooting can be performed using Extended Events
  - More advanced scenarios will be covered by the forthcoming Advanced Extended Events course