Auditing in SQL Server 2008

SQL Server Technical Article

Writer: Il-Sung Lee, Art Rask

Technical Reviewer: Jack Richins, Rick Byham, Sameer Tejani, Al Comeau, JC Cannon

Published: February 2009

Applies to: SQL Server 2008

Summary: With SQL Server Audit, SQL Server 2008 introduces an important new feature that provides a true auditing solution for enterprise customers. While SQL Trace can be used to satisfy many auditing needs, SQL Server Audit offers a number of attractive advantages that may help DBAs more easily achieve their goals such as meeting regulatory compliance requirements. These include the ability to provide centralized storage of audit logs and integration with System Center, as well as noticeably better performance. Perhaps most significantly, SQL Server Audit permits fine-grained auditing whereby an audit can be targeted to specific actions by a principal against a particular object. This paper provides a comprehensive description of the new feature along with usage guidance and then provides some practical examples.

Introduction

A key part of any data security strategy is the ability to track who has accessed, or attempted to access, your data. This provides the ability to detect unauthorized access attempts or, if necessary, to piece together the actions of malicious insiders who misused their legitimate access. Furthermore, a rich and robust tracking capability can provide oversight of sensitive configuration changes made by administrators.

Such considerations are ever more relevant in today's information economy. Data is collected, stored, used, and misused at an ever increasing rate. Governments and private sector organizations around the world are responding to this by establishing various compliance regimes to improve the stewardship of data by those who hold it. A few of the most widely known examples include:

- European Union Data Protection Directive, a strict data protection policy with implications across the EU and the global economy.
- HIPAA, or Health Insurance Portability and Accountability Act, part of United States law
- Sarbanes-Oxley, part of United States law governing corporations.
- Payment Card Industry Data Security Standard, mandated by major credit card companies, with worldwide implications.

These formal regulations affect organizations of all sizes, in all industries, around the world. They place significant pressure on organizations to ensure their IT platforms and practices are compliant. And ultimately, these requirements land at the feet of the DBAs, developers, and IT professionals who manage the data.

It is important that a data management platform provide the means to meet these requirements, and do so efficiently. To address these needs, SQL Server 2008 introduces a rich and deeply integrated auditing capability that offers major improvements over previous versions of the Microsoft® SQL Server® database software.

This paper will review the new audit features of SQL Server 2008, compare them to past versions, and walk through some implementation examples.

Auditing in SQL Server 2005

In SQL Server 2005 and earlier versions, auditing was implemented using a combination of tools. At the server

level, options were available to log successful and/or failed logins to the Application log in the Windows® operating system and the SQL Server Error Log. Login triggers, server triggers, and DDL triggers were available to do custom auditing of specific types of events. The tool of choice for detailed activity audits, however, was SQL Trace. SQL Trace is a mechanism for monitoring a broad range of events internal to the SQL Server database engine. It is used for detecting deadlocks, monitoring application performance, debugging, and many other development or administrative purposes. It is useful as an audit tool because it can capture individual statements as they are executed, including the user id.

SQL Trace has limitations as an auditing tool, though. Configuring and managing traces requires the use of a separate tool, the SQL Server Profiler. This is a tool well-loved by many developers and administrators, but it is not integrated with SQL Server Management Studio, the primary SQL Server administrative interface. And, its interface is not focused on the specific task of creating and managing information audits. At the Transact-SQL level, traces are configured by a set of system stored procedures which take opaque numeric codes as arguments – not the most inviting of management APIs.

Trace functionality in SQL Server 2005 is also accessible programmatically via Server Management Objects (SMO). However, the SMO trace classes are dependent on binary trace definition files (.tdf) authored by SQL Server Profiler, so they do not really serve as a standalone API for managing server audits.

Finally, because SQL Trace is a multipurpose tool, it doesn't always easily answer the questions an auditor would care about. When used as an audit tool, SQL Trace captures statements. But this does not necessarily expose at a glance what information a user read or modified. Statements might reference views, stored procedures, or user-defined functions, in which case further analysis is required when the real question is "who read data from this table?"

All the tools described above remain in SQL Server 2008, and they will continue to be useful for a wide array of needs. But SQL Server 2008 brings a new, more focused, and more deeply integrated auditing capability to the table.

Auditing in SQL Server 2008

SQL Server 2008 Enterprise enhances the auditing capabilities of the server by introducing native auditing functionality into the database engine. The new SQL Server Audit feature maintains all the capabilities of the SQL Server 2005 auditing solutions and provides enhancements such as flexibility in audit data targets and granular auditing.

Overview

The SQL Server Audit feature in SQL Server 2008 is intended to replace SQL Trace as the preferred auditing solution. SQL Server Audit is meant to provide full auditing capabilities and only auditing capabilities, unlike SQL Trace which also serves for performance debugging. As a result, SQL Server Audit was designed with the following primary goals in mind:

- Security – The audit feature, and its objects, must be truly secure.
- Performance - Performance impact must be minimized.
- Management – The audit feature must be easy to manage.
- Discoverability - Audit-centric questions must be easy to answer.

The subsequent sections discuss how each one of these design goals is realized.

As mentioned earlier, SQL Server Audit is now native to the server. Doing so affords the new SQL Server Audit feature performance benefits and allows the audit objects to be treated as first class database objects. Being a first class object means that the audit objects can be created and manipulated through Transact-SQL DDL statements (or through the SQL Server Management Studio if you prefer). Furthermore, the audit objects will be secured through the database's familiar permission model.

When an audit object is created, the destination for audit events must be specified. The typical target is a file which is probably suitable for most cases. However, in SQL Server 2008, the audit events can also be written to the Windows Application log. This facilitates access to the audit information in some situations and can allow consolidation through management applications. In addition, the very security conscious will be pleased to learn that the Windows Security log can be specified as a target for Audit (starting with the Windows Server® 2003 operating system). This is significant in that the Windows Security log is considered to be resistant to tampering and nonrepudiation. Another benefit of using the Security log is that the Audit Collection Service (ACS) of the Microsoft System Center Operations Manager can be used to securely collect audit information from the security logs of multiple machines and generate consolidated reports.

A final, but important, point to mention about SQL Server Audit in SQL Server 2008 is the granularity at which the audit can be specified. The auditing of activity of users, roles, or groups on database objects can be restricted

down to the table level. That is, you can target SQL Server Audit to track specific activities of a user or users down to the individual table level. For example, SQL Server Audit allows a record to be made of all the UPDATEs to the **Payroll** table by DBO.

Security

Auditing is a central component of a database security strategy and as such the SQL Server Audit feature itself must be highly secure and securable. Because it is a database object, the privilege to create, delete, or modify an audit object is controlled by the database engine's permission model and enforcement control. The SQL Server Audit feature introduces a new server-level permission called ALTER ANY SERVER AUDIT to allow a principal to CREATE, ALTER, or DROP an Audit or Server Audit Specification object. Likewise, a database-level permission called ALTER ANY DATABASE AUDIT is introduced to allow a principal to CREATE, ALTER, or DROP a Database Audit Specification object. An audit specification can briefly be described as an object that describes the activity to capture in the audit (a more thorough description is found later in this paper).

The audit log is obviously another asset that needs protection. Careful consideration must be given regarding where to send the audit data. Ideally, the audit information should be sent to a location that cannot be modified or tampered with, even by a sysadmin. A simple strategy to accomplish this is to send the audit events to a file on a share to which the SQL Server service account only has write access. The person in the auditor capacity would obviously need read access to this file but no one else. Also, if the share is on a different computer, it's possible to obtain protection from the Windows administrator (of the SQL Server computer) as well, although there are obviously some performance considerations and vulnerability to network outages. Another option is the Windows Security log. This is a good choice for someone who wants to store the SQL Server Audit log locally but wants protection from the sysadmin and even the local Windows administrator. One final point – SQL Server does not encrypt or otherwise protected the Audit log. Such additional layers of protection can be applied, of course, through a variety of means external to SQL Server, such as file system encryption.

If, for whatever reason, SQL Server Audit is unable to write its audit events to the audit target, you can configure the audit object to shut down the server instance. While shutting down the server instance may seem drastic, it is a necessity for certain scenarios, such as meeting the requirements of the Common Criteria to ensure that the server cannot operate without its activity being audited. To configure the audit to behave in such a manner, the person issuing the CREATE or ALTER AUDIT DDL must additionally have the SERVER SHUTDOWN server permission. If the server cannot start because of SQL Server Audit, it is possible to bring the server up by starting SQL Server with the **-m** trace flag; SQL Server Audit itself will not be deactivated and only the shutdown behavior will be disabled. Please note that **-m** starts SQL Server in single-user mode and is meant to allow a DBA to start the database and make changes to the SQL Server Audit configuration if necessary. Alternatively, the **-f** trace flag can be used to start SQL Server in minimal configuration mode, in which case Audit will be disabled but Audit DDL can still be issued. In practice, **-m** should be sufficient for most cases that require SQL Serve Audit reconfiguration.

If a failure to write the Audit event does not trigger the SQL Server instance to shut down, what happens with the audit records? The answer is that Audit events are buffered in memory until they can be flushed to the target. If the records fill the memory buffer and cannot be written to the Audit log, the server blocks any new activity that would result in an audit event being written until the buffer space is freed up or the audit is disabled. The size of the memory buffer varies, but it is around 4 MB per audit in the default case, which can accommodate at least 170 audit events (the exact number depends upon the amount of data contained in each event). If the problem is not correct before the operating system returns an error, such as a disk write failure, the Audit session is taken offline with a corresponding error written to the server's error log; all buffered and new audit events are discarded. Upon correction of the problem, the audit object will need to be restarted in order for auditing to resume. If lost audit records are unacceptable, the audit should be configured to shut down rather than continue upon write failure. To ensure tight synchronicity between the events captured in the audit log and the activity on the server, SQL Server Audit can be configured to write the audit entries to the log in a synchronous fashion, meaning that transactions are blocked until the event is written to its destination. The tradeoff here is obviously that performance may be affected adversely. In most situations, asynchronous Audit log writing is recommended. As described below, the length of time before writes occur can be configured; increasing the time value from the default of one second can improve the performance of the server.

One last point to highlight is that any changes to the state of the server audit object itself are recorded in its audit log. This is a shortcoming of SQL Trace, and it is an important feature because it can potentially capture any attempts to circumvent the audit. Assuming that the audit events are sent to a target that cannot be modified by the SQL Server service account, this feature can effectively protect the audit log from the DBA. Furthermore, it is possible to configure SQL Server Audit to record changes to all audit objects defined within the instance, not just the immediate server audit object (this can accomplished by adding the AUDIT_CHANGE_GROUP to the server audit specification; this will become more clear later in this document).

Performance

SQL Server 2008 introduces a new high-performance eventing infrastructure called SQL Server Extended Events. The SQL Server Audit feature is built on top of Extended Events to leverage the performance benefits and provide

both asynchronous and synchronous write capabilities (by default, SQL Server Audit uses the asynchronous eventing model). One thing to note is that the Audit event is a protected type of Extended Event so the CREATE/ALTER/DROP SESSION EVENT commands cannot be used to manipulate SQL Server Audit directly. For more information about Extended Events, see SQL Server Books Online.

By default, the audit events are written to the audit target in an asynchronous fashion for performance reasons. When tighter guarantees of audit records being written to the audit log are required, you can select synchronous write, with the understanding that some amount of performance degradation should be expected (the exact amount depends on the amount and type of audit records generated). The choice of asynchronous or synchronous is controlled by the QUEUE_DELAY option of the CREATE AUDIT DDL (explained below); by default, the value is 1000, which indicates that SQL Server Audit will queue the audit record for up to one second before writing to the target. If tolerances allow, you can set this value to a larger number, which can increase performance; the tradeoff is that activity recorded in the log might be increasingly out of sync with the actual database activity, and there is more potential for lost records if a fatal error occurs on the server. To enable synchronous logging of SQL Server Audit, set the QUEUE_DELAY to the value 0 (zero).

Targeted auditing can also aid in minimizing the performance impact. The obvious strategy is to limit auditing to the exact set of actions of interest and nothing more. The fine-grained auditing capabilities of SQL Server Audit allow auditing to be configured so that it targets specific actions, principals, and objects (down to the individual table level). As a result, resources are not wasted generating unwanted audit information. This is in contrast to SQL Trace, which typically generates a lot of records and relies upon post-filtering to provide targeted auditing.

Because of the architecture of SQL Server Audit, the best performance is generally achieved by using a file as the audit target. However, the performance difference between the different audit targets is highly dependent upon the audit configuration, among other things.

Management

The audit features of SQL Server 2008 are fully manageable using the main management interfaces of the product, specifically SQL Server Management Studio, Server Management Objects (SMO), and Transact-SQL DDL.

In SQL Server Management Studio, server level audit configuration is available in the user interface under the Security folder. The two relevant subfolders are Audits and Server Audit Specifications. Database-level audit configuration is available under the Security folder of each database, in the Database Audit Specification subfolder.

In contrast to previous versions, the audit configuration can be fully managed programmatically via SMO, using objects in the Microsoft.SqlServer.Management.SMO namespace.

Finally, rather than the system stored procedures used to configure SQL Trace, SQL Server Audit uses clear SQL DDL syntax to create, manage, and secure audits.

Discoverability

Audit data can be written to a binary file or to the Windows Event log (Application or Security). When written to a file, the audit data is accessible through a built-in table-valued function, which allows the use of regular SELECT syntax to query the audit trail. Events written to the Event log can be accessed with Event Viewer or with specialized Event log management software, such as Microsoft Systems Center Operations Manager. Accessing and analyzing audit data is discussed in more detail later in this paper.

Audit Value of Data

Besides providing a robust infrastructure for collecting audit data, SQL Server 2008 makes it easier to meet specific audit requirements. Audit records are captured based on permission checks on database objects. This means, for example, that a SELECT on table **A** is recorded as such, regardless of whether the actual SQL statement referenced table **A**, a view, a stored procedure, or another object.

Also, SQL Server 2008 allows granular definition of audit criteria. Audits can be scoped to individual tables, to specific DML actions (for example, DELETE vs. SELECT) and to specific principals. This granularity can reduce the volume of audit data that must be stored and analyzed to meet a specific set of requirements.

Technical Overview of Auditing

To understand the setup and management of auditing in SQL Server 2008, it is best to start with the three main objects which describe audits.

The **Server Audit** object describes the target for audit data, plus some top-level configuration settings. Think of a Server Audit object as the declaration of the audit sink, or destination. This destination can be a file, the Windows

Application log, or the Windows Security log. The allowable delay for writing events to the destination can also be configured on this object. Note that the Server Audit object contains no information about what is being audited – just *where* the audit data is going. Multiple Server Audit objects can be defined with each object being specified and operational independent from one another.

The **Server Audit Specification** object describes *what* to audit. As its name suggests, this object is focused on server instance-wide actions. A server audit specification is associated with a server audit in order to define where the audit data is written. There is a one-to-one relationship between the Server Audit Specification object and the Server Audit object.

The **Database Audit Specification** also describes *what* to audit. But, as its name suggests, this object is focused on actions which occur in a specific database. Where the audit data is written is defined by the association of a database audit specification with a Server Audit object. Each database audit specification can be associated with only one server audit. A server audit, for its part, can be associated with only one database audit specification per database.

The interaction of these objects to define an overall audit regime is depicted in Figure 1 below.

All the relationships between objects in Figure 1 are optional. Of course, without a minimum of one server audit and one specification (server or database), an audit cannot produce any data. All of these objects – Server Audit, Server Audit Specification, and Database Audit Specification – can be fully managed with DDL, the SQL Server Management Studio administrative interface, or SMO.

Let's drill down on the details of each of these parts of an audit definition.

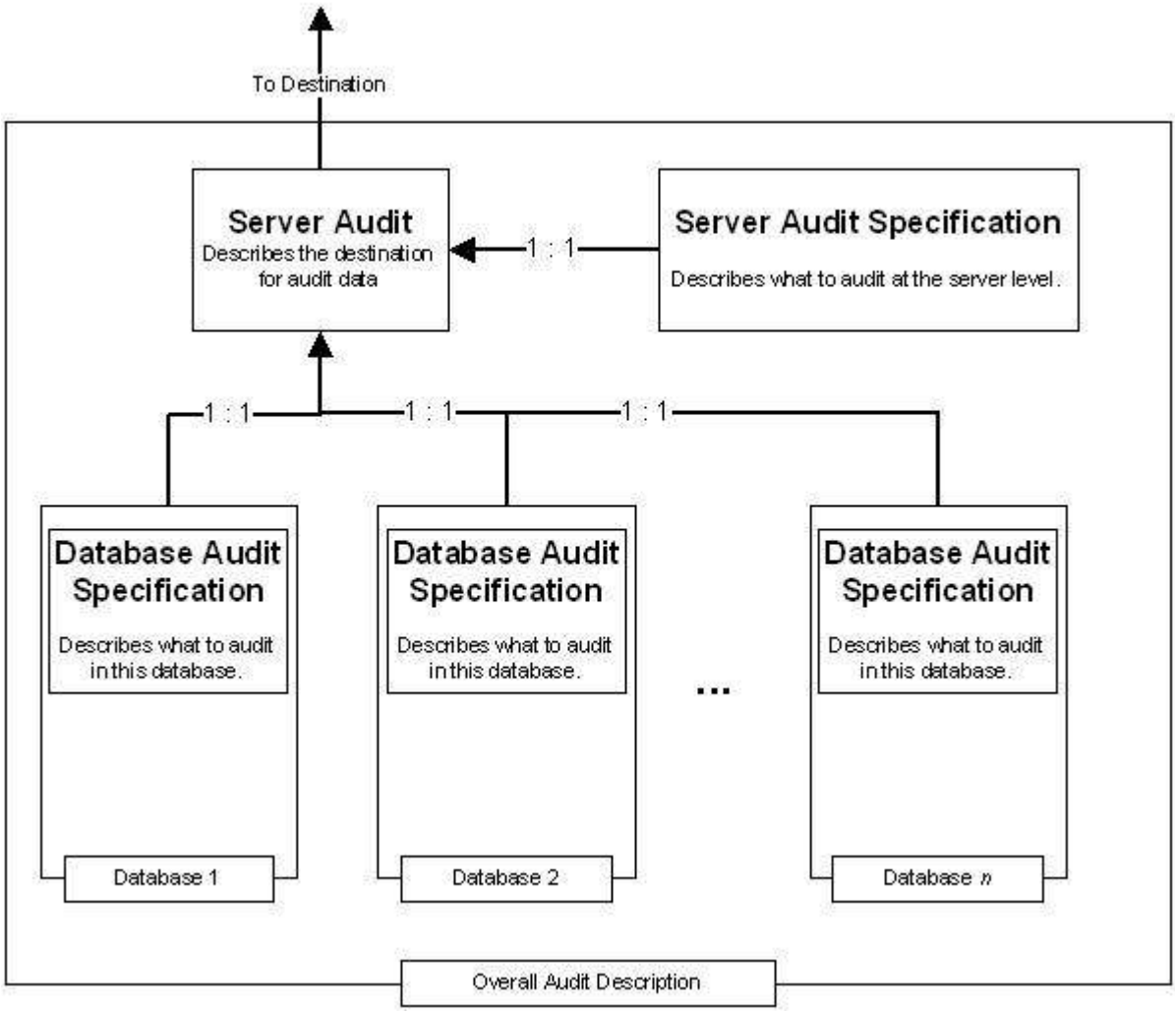

Figure 1: Audit object layout

Server Audit

Server audits are created with the CREATE SERVER AUDIT statement. Besides a name, the most important thing

specified when the server audit is created is the destination for the audit data. In SQL Server 2008, the available destinations are a binary file, the Windows Application log, and the Windows Security log (specific permissions – discussed later – are required of the SQL Server service account in order to write to the Security log). For file destinations, the user must provide at least the file path and optionally, the maximum size and the maximum number of rollover files. SQL Server names audit files automatically, according to the following pattern:

<audit_name>_<audit_guid>_nn_<timestamp_as_bigint>.sqlaudit

When the active audit file reaches the maximum size specified, it automatically rolls over to a new file. This process continues up to the maximum number of rollover files specified by the MAX_ROLLOVER_FILES argument. When or if the maximum number of rollover files is reached, SQL Server begins deleting an existing audit file for every new one created, starting with the oldest first. Note that if the delete operation fails for whatever reason, the audit silently continues.

Optionally, the audit can reserve the maximum disk space on startup. Naturally, the SQL Server service account must have write permissions on the destination folder.

To provide control over security-performance tradeoffs, a queue delay can be configured on the server audit. If this value is 0, audit records are written synchronously with the action that generated them. Alternatively, a queue delay in milliseconds can be specified, to lessen performance impact under load. The minimum delay is 1,000 milliseconds (the default).

To meet the most stringent security requirements, a server audit optionally can be configured so that SQL Server will shut down if the audit fails. For example, if the destination drive for the audit log becomes unavailable, SQL Server shuts down and can be restarted manually with the **–m** command switch unless the root cause of the shutdown is corrected.

By default, server audits are created in a disabled state. This can be overridden in DDL so that the audit is enabled immediately. In most cases, however, administrators will want to fully configure the audit before enabling it.

A server audit is automatically tagged with a GUID which uniquely identifies it. To support distributed scenarios such as database mirroring, the CREATE SERVER AUDIT statement allows this GUID to be assigned explicitly. After the audit is created, the GUID cannot be changed.

The following DDL example shows the creation of an audit named PCI_Audit, which will write to a file with no delay, and force the server to shut down in the event of a failure to write to the file.

Transact-SQL Copy Code

```
CREATE SERVER AUDIT PCI_Audit
TO FILE (FILEPATH = 'F:\AuditLogs\', MAXSIZE = 1GB, MAX_ROLLOVER_FILES = 80)
WITH (QUEUE_DELAY = 0, ON_FAILURE = SHUTDOWN)
```

This server audit is disabled initially.

The server audit can be modified after creation by using the ALTER SERVER AUDIT statement. The following example changes the destination to the Windows Application log and relaxes the queue delay and failure settings.

Transact-SQL Copy Code

```
ALTER SERVER AUDIT PCI_Audit
TO APPLICATION_LOG
WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
```

The ALTER SERVER AUDIT statement is also used to enable or disable a server audit.

Transact-SQL Copy Code

```
ALTER SERVER AUDIT PCI_Audit
WITH (STATE = ON)
```

A server audit can only be enabled or disabled outside of the scope of an active transaction.

In SQL Server Management Studio, server audits are created and managed in the Audits folder, which is under the server-level Security folder.

Figure 2: Creating an audit in SQL Server Management Studio

Permissions

Actions on a server audit (CREATE, ALTER, or DROP) require the ALTER ANY SERVER AUDIT permission, which is covered by the CONTROL SERVER permission.

Writing to the Windows Security Log

Writing to the Windows Security log deserves special mention, because there are some extra considerations to take into account. Because writing to the Windows Security log is a very privileged operation, the SQL Server service account needs to be granted the SeAuditPrivilege (also known as "Generate Security Audit") privilege. This can be accomplished though the Local Security Policy tool (secpol.msc). Just add the service account to *LocalPolicies/User Rights Assignment/Generate Security Audits*.

In addition, the audit policy granularity for Windows must be modified before the SQL Server entries can be logged. On Windows Server 2003, use the Security Policy Tool to modify the *Local Policies/Audit Policy/Audit Object Access* setting; both 'Success' and 'Failure' should be enabled. On the Windows Vista® operating system and Windows Server 2008, use the auditpol command-line tool and issue the command auditpol /set /subcategory:"application generated" /success:enable /failure:enable. Please note that in some domain environments, the domain policy may override these settings, and assistance from the Domain Administrator will be required in this case. Users who install SQL Server 2008 Developer onto a Windows XP computer will not be able to write the audit information to the Windows Security log because this capability is not supported by the operating system.

Server Audit Specification

Server audit specifications can be created with the CREATE SERVER AUDIT SPECIFICATION statement. A server audit must exist before a server audit specification can be created, since the DDL requires that an audit name be specified.

The main content of a server audit specification is the list of action groups that it will audit. Auditable actions are collected into groups and cover server-level activity such as server configuration changes and login/logout. These groups are hardwired, and they are detailed in the SQL Server documentation.

The following statement creates a server audit specification to track backup/restore events, SQL Server service starts and stops, and changes in the membership of server roles.

Transact-SQL Copy Code

```
CREATE SERVER AUDIT SPECIFICATION PCI_Server_Mgmt_Spec
FOR SERVER AUDIT PCI_Audit
  ADD (BACKUP_RESTORE_GROUP),
  ADD (SERVER_STATE_CHANGE_GROUP),
  ADD (SERVER_ROLE_MEMBERSHIP_CHANGE_GROUP)
WITH (STATE = OFF)
```

Server audit specifications can be reassigned to a different server audit after they are created by using the ALTER SERVER AUDIT SPECIFICATION statement. This statement also allows the list of audited action groups to be

changed and the state to be toggled between enabled (ON) and disabled (OFF).

The following example adds an action group to the specification created in the previous statement.

Transact-SQL Copy Code

```
ALTER SERVER AUDIT SPECIFICATION PCI_Server_Mgmt_Spec
  ADD (SERVER_PRINCIPAL_CHANGE_GROUP)
```

Change to the state of an audit (enabled or disabled) can only be made outside the scope of a transaction. Also, if the ALTER SERVER AUDIT statement changes the state from ON to OFF, it may not include any other changes within the same statement.

In SQL Server Management Studio, server audit specifications are created and managed in the Server Audit Specifications folder, which is under the server-level Security folder.

Permissions

Actions on a server audit specification (CREATE, ALTER, or DROP) require the ALTER ANY SERVER AUDIT permission, which is covered by CONTROL SERVER.

Database Audit Specification

Database audit specifications are much like server audit specifications, but there are some significant differences. As the name implies, of course, database audit specifications pertain to the auditing of database level actions. As with server audit specifications, a predefined list of audit action groups can be used to define what the specification tracks.

In addition to this, however, database audit specifications allow more specific audit actions to be tracked. This is where granular actions within the database can easily be targeted. Before exploring the details of this, examine the following statement.

Transact-SQL Copy Code

```
CREATE DATABASE AUDIT SPECIFICATION PCI_Txn_Database_Spec
FOR SERVER AUDIT PCI_Audit
  ADD (DATABASE_OBJECT_PERMISSION_CHANGE_GROUP),
  ADD (SELECT ON dbo.Customer BY dbo),
  ADD (INSERT ON dbo.Customer BY dbo),
  ADD (UPDATE ON dbo.Customer BY dbo),
  ADD (DELETE ON dbo.Customer BY dbo),
  ADD (EXECUTE ON dbo.usp_SubmitPO BY public)
```

This specification tracks all actions that fall into the DATABASE_OBJECT_PERMISSION_CHANGE group, which is a useful thing to monitor if you have multiple administrators working in your database. In addition, it also tracks specific DML actions against the **Customer** table by the dbo user, and any execution of the **usp_SubmitPO** stored procedure. Also note the use of the keyword public to track changes by all users.

In SQL Server 2008, this granular action definition is a key distinction of database audit specifications from their server-level counterparts. The general form for defining an action to be audited is:

<action> ON <object> BY <principal>

Auditing occurs regardless of the means by which the action was taken. In other words, a SELECT on the **Customer** table will be audited if the user action was a SELECT statement on **Customer**, a SELECT on a view that queries the table, an EXECUTE of a stored procedure that queries the table, and so on.

Note that <object> in the above format can refer to an object like a table, view, or stored procedure, or an entire database or schema. For the latter cases, the forms DATABASE::<db_name> and SCHEMA::<schema_name> are used, respectively.

Specifying a principal allows the audit to be narrowly focused, thereby minimizing the amount of data captured in the audit. In order to audit an action for all principals, the public role can be specified. For example, the following action records all SELECTs on all objects within the MyDB database by any user in the MyDBRole role.

Transact-SQL Copy Code

```
SELECT ON DATABASE::MyDB BY MyDBRole
```

As with server-level specifications, a database audit specification can be enabled or disabled at creation, or afterwards with the ALTER DATABASE SERVER SPECIFICATION statement. The ALTER statement also allows modification of the list of audited actions and action groups.

In SQL Server Management Studio, database audit specifications are created and managed in the Database Audit Specifications folder, which is under the database-level Security folder for the database in question.

Permissions

Actions on a database audit (CREATE, ALTER, or DROP) require the ALTER ANY DATABASE AUDIT permission, which is covered by CONTROL DATABASE.

More on the Technical Architecture of Audit

Before we move into how audit data is read and used, a few more technical aspects of the implementation are worth mentioning. Earlier, it was stated that SQL Server Audit is native to the server. There are many aspects of this feature that are more tightly bound to the DBMS, not the least of which is the way in which audit events are triggered. Auditing in SQL Server 2008 is implemented by hooking the internal permissions checks, which occur inside the database engine. When a permission check occurs, SQL Server Audit has the opportunity to produce an audit record, if a relevant audit is specified. This means all forms of access to an object are audited.

A second aspect of SQL Server Audit that should be noted is its behavior within transactions. There are two pieces to this. The first is how operations on an audit itself interact with transactional boundaries. All audit-related DDL statements, including the CREATE SERVER AUDIT statement, can be issued within an active transaction unless they trigger changes in the state of an audit object. If within a transaction, the creation of the audit will be scoped within the transaction – it will commit or roll back with the transaction.

Another piece is how auditable events are recorded when they occur within a transaction. The simple way to describe this is: They are always audited. Clearly there would be negative security implications if data could be read, but no audit was recorded because a pending transaction was rolled back. So, the recording of audit events occurs irrespective of transaction scope and whether a rollback occurs or not.

Using Audit Data

This section includes information about the ways in which you can use and apply the data generated by SQL Server Audit.

Audit Files

When a SQL Server 2008 server audit is configured with the audit destination of File option, the audit data is written to a binary file on disk. The target location can be a local folder or a UNC path to a network share.

Although the server audit can specify the path where the audit file will be created, the file name is automatically generated by SQL Server. The file name pattern is:

<audit_name>_<audit_guid>_*nn*_<timestamp_as_bigint>.sqlaudit

In this pattern, *nn* is a sequential partition number that is incremented as new audit files are generated, if the audit is configured to allow rollover to new files.

The audit file may be written to a folder protected by NTFS encryption, provided the SQL Server service account has the appropriate permissions. Writing to a folder with NTFS compression is also permitted. In both cases, of course, some performance impact should be expected. If the target location for the audit file is a remote file share, the audit content is not encrypted when travelling across the network.

As mentioned, the audit file is in a binary format and is not readable with a text editor. SQL Server provides a table-valued function called fn_get_audit_file(), which is used to read audit data. This function takes the arguments indicated in Table 1.

Parameter name	Type	Description
File_pattern	**nvarchar**	The directory or path and file name indicating the location and audit file set to be read. A single asterisk is allowed as a wild card character. Can be a local or UNC path.

Initial_file_name	**nvarchar**	Specifies the path and file name of the file in the file set from which to start reading.
Audit_file_offset	**bigint**	A known location within the file in the initial_file_name parameter. Reading starts from the next record after this byte offset.

Table 1: Arguments to fn_get_audit_file()

The following statement reads all audit files for a specific audit.

Transact-SQL Copy Code

```
SELECT * FROM fn_get_audit_file(
'D:\Audits\MyAudit-_C26128D1-F97B-4B82-9E47-B6A296045B05_*.sqlaudit',
default, default)
```

The following example reads all audit files in the E:\SqlAudits\ folder.

Transact-SQL Copy Code

```
SELECT * FROM fn_get_audit_file('E:\SqlAudits\*', default, default)
```

And this example reads all audit records in a specific audit, starting from a known offset.

Transact-SQL Copy Code

```
SELECT * FROM fn_get_audit_file(
'D:\Audits\MyAudit-_C26128D1-F97B-4B82-9E47-B6A296045B05_*.sqlaudit',
'D:\Audits\MyAudit-_C26128D1-F97B-4B82-9E47-B6A296045B05_0_128439520854140000.sqlaudit',
5120)
```

Incidentally, the initial_file_name and audit_file_offset parameters can be determined by looking at the file_name and audit_file_offset columns respectively from the output of the last invocation of fn_get_audit_file().

Naturally, because this is a SELECT from a table-valued function, a WHERE clause, ORDER BY clause, and other standard SQL SELECT syntax can be used to shape the result set.

Because the fn_get_audit_file() function takes an explicit path as an argument, it is possible to move inactive audit files to a new location and read them using this function. At this point the audit file is just static data. Thus audit files can be moved and queried at will after they are no longer receiving audit events. A useful implication of this is that audit files from multiple servers can be consolidated in a single folder and queried as a set. In this way, administrators or auditors can easily look for key events across a distributed set of SQL Servers.

The fn_get_audit_file() function returns a rich set of information captured by SQL Server Audit. More than two dozen columns are available, providing details in several areas, such as:

- The date and time of the event, and whether it succeeded (or permissions were denied)
- The user context of the event (principal id and name at server and database level)
- The type of action (for example, SELECT, EXECUTE, DELETE)
- The target object
- The connection context (server and instance, database, schema)
- The actual Transact-SQL statement that triggered the audit event (if applicable)
- The audit file name and the position of the audit record in the file
- Additional XML information for some special events, such as audit management events

As is apparent, a great deal of information is available about each audited event. One of the columns, **statement**, contains the actual text of the action that caused the audited event, although the actual data values in the statement might not be displayed if the statement was parameterized. Note that this may not directly name the audited resource. For example, an audited SELECT event on **Table1** might be caused by a SELECT on View1, which references **Table1**.

The **statement** field is **nvarchar** and is 4,000 characters in length. In the event that a statement exceeds this size, SQL Server 2008 generates multiple audit records, spreading the statement text across as many records as necessary to capture the full content. A sequence number field in the audit records enables the full statement text to be reassembled.

The fn_get_audit_file() function is your jumping-off point to implement the best audit analysis solution for your environment. Because it essentially presents the audit data as a table, it provides the means to query, mine, archive, or report on the audit logs in any way necessary. In this sense, it is perhaps the most flexible of the three possible targets for audit data.

The Windows Event Log

A SQL Server 2008 audit can alternatively be configured to log to the Windows Application log or the Windows Security log. This can be valuable for a few reasons:

- The Windows event log is a familiar location and format for many server administrators and auditors.
- The Security log has inherent restrictions on writing, tampering, and viewing, which make it attractive as a repository for audit information.
- There are existing Microsoft and third-party solutions aimed at consolidating, monitoring, and alerting on Windows event logs. For example, Microsoft System Center has a component called Audit Collection Services that can selectively consolidate Security event logs from many servers. And the System Center Operations Manager product is dedicated to monitoring, reporting, and alerting on Windows event logs.

When a server audit is configured with the TO APPLICATION_LOG or TO SECURITY_LOG option, all audit events are written to the relevant log. The events contain the same information as is returned by the fn_get_audit() table-valued function for audits to file. In the event log, this information is written to the event body in text format.

For users who intend to set up monitoring or alerting on event log entries, it is important to know what event log specific attributes will be assigned to SQL Server Audit entries. All SQL Server Audit events are logged with the SQL Server instance name as the event source. The event id is 33205. The description of the event is where distinguishing characteristics of events can be found. So, to implement a filtering or alerting on audit events, the Event log management tool used will need to support identification of events based on the description for example, regular expression searches).

Performance Comparison

Using a sample of OLTP workloads typical of a customer environment, the performance of SQL Server Audit was measured and compared to SQL Trace where possible. The results consistently indicated a performance advantage for SQL Server Audit with improvements ranging from 11% to 45%. The following list provides a short profile of the workloads used and the subsequent table reveals values for a sample of the run times for each workload. For the comparison, SQL Trace is running in C2 Audit mode and SQL Server Audit is configured to audit the equivalent events and write the output to a file.

Workload 1 contains 11 databases, ranging from 1.94 MB to 1812.5 MB. It also has 755 tables with an average of 2761 rows, with primarily **char** and then **smallint** and **decimal** columns.

Here is the activity summary for workload 1 (% of workload):

- Stored procedures:
    - **sp_cursorfetch**
- EXECUTE (38.72%)
- Statements:
    - ORDER BY (21.90%)
    - SELECT FROM (39.51%)
- 1,219,234 statements were executed.

Workload 2 contains 2 databases ranging from 64 MB to 423.88 MB. It has 35 tables with an average of 49,141 rows, with primarily **bit**, **varbinary**, **int**, and **nvarchar** columns.

Here is the activity summary for workload 2 (% of workload):

- EXECUTE (100%)
- 1,633,557 statements were executed

Workload 3 contains 3 databases ranging from 1.94 MB to 1059.63 MB. It has 154 tables with an average of 586 rows, with primarily **real**, **numeric**, and **float** columns.

Here is the activity summary for workload 3:

- Stored procedures:
  - **sp_execute**
  - **sp_prepare**
  - **sp_unprepare**
- EXECUTE (94.25%)
- Statements:
  - SELECT FROM (39.87%)
- 585,400 statements were executed.

Workload 4 contains 1 database at 3235.75 MB. It has 84 tables with an average of 144,245 rows, with primarily **int**, **varchar**, and **datetime** columns.

Here is the activity summary for workload 4 (% of workload):

- Stored procedures:
  - **sp_cursorfetch**
- EXECUTE (95.89%)
- 3,435,303 statements were executed.

Workload 5 contains 1 database at 174.94 MB. It has 152 tables with an average of 4,108 rows, with primarily **char** and **numeric**, and then **int** and **float** columns**.**

Here is the activity summary for workload 5 (% of workload):

- Stored procedures:
  - **sp_cursorclose**
  - **sp_cursorfetch**
  - **sp_cursoropen**
- EXECUTE (78.58%)
- Statements:
  - SELECT FROM (26.96%)
- 296,642 statements were executed.

Workload	Base Time (min)	SQL Trace (min)	SQL Server Audit (min)	Perf Cost Vs. Base	Perf Gain vs. Trace
1	13.30	15.89	14.13	6.24%	11.08%
2	41.30	101.85	55.93	35.42%	45.09 %
3	5.09	6.29	5.57	9.43%	11.45%
4	63.36	76.64	68.13	7.53%	11.10%
5	3.59	4.76	4.00	11.42%	15.97%

Table 2: Performance numbers for SQL Server Audit vs. SQL Trace

In the table, the PERF COST VS BASE is calculated as (SQL AUDIT– BASE TIME) / BASE TIME, and the PERF GAIN VS TRACE is calculated as (SQL TRACE – SQL AUDIT) / SQL TRACE.

The improvement between SQL Trace and SQL Server Audit ranges from 11.08% to 45.09%, with Workload 2 showing the greatest difference between the two. This is likely a result of the workload generating a huge amount of Audit events which tends to emphasize SQL Server Audit's greater efficiency at file I/O. Also observed from the table is that the cost of running SQL Server Audit ranges from 6.24% to 35.42%. Keep in mind that these values

are with a large number of audit action groups enabled so the cost would be lower with more modest Audit settings. Since SQL Trace lacks the fine-grained auditing capabilities of SQL Server Audit, a performance comparison between the two is not possible for this scenario.

Examples

The following examples provide real-world scenarios for the use of SQL Server Audit.

Example 1

A relatively simple but very relevant usage of SQL Server Audit is to monitor the activity of privileged users accessing a sensitive table. This can easily be accomplished with the SQL Server Audit feature in SQL Server 2008 by setting up the proper database audit specification. For this example, assume that the sensitive table is in the **hr_db** database and is called **hr.salary** and we want to detect when the dbo (and consequently sysadmin) tries to query the table. In this example, the audit information will be written to a file on a remote share called \\AuditServer\Audit, on which the SQL Server service instance has permissions to write but not read or modify.

First we need to create the Audit and the Database Audit Specification objects.

Transact-SQL Copy Code

```
USE master
CREATE SERVER AUDIT audit1 TO FILE (FILEPATH='\\AuditServer\Audit')
USE hr_db
CREATE DATABASE AUDIT SPECIFICATION hr_dbspec FOR SERVER AUDIT audit1
ADD(SELECT,UPDATE,INSERT,DELETE ON hr.salary by dbo)
```

Because the objects are created disabled by default, they need to both be enabled.

Transact-SQL Copy Code

```
USE master
ALTER SERVER AUDIT audit1 WITH (STATE=ON)
USE hr_db
ALTER DATABASE AUDIT SPECIFICATION hr_dbspec WITH (STATE=ON)
```

At this point, the audit is turned on and all queries against the **hr_db** table by the dbo or the sysadmin are recorded, as are any actions that enable or disable the audit.

To confirm that both the audit and the audit specification are enabled, the system views can be queried.

Transact-SQL Copy Code

```
SELECT is_state_enabled FROM sys.server_file_audits
SELECT is_state_enabled FROM sys.database_audit_specifications
```

Example 2

Some compliance regulations require that access and usage of cryptographic keys be audited. SQL Server Audit is capable of recording all activity that touches a symmetric encryption key or the private key of a certificate or an asymmetric key pair. Concisely, this activity is covered by the DATABASE_OBJECT_CHANGE_GROUP action group at both the server and database levels. The DATABASE_OBJECT_CHANGE_GROUP action group covers all activity involving key usage as well as CREATE, ALTER, and DROP of objects in a database not contained within a schema. Aside from the initial setup period of the database, the incidents of CREATE, ALTER, and DROP should be low, meaning that most of the activity captured under this action group should be restricted to cryptographic key activity. For this example, creation of the Audit will be demonstrated using SQL Server Management Studio. To create the audit object, expand the Security tree under the instance, right-click **Audits,** and then click **New Audit**. The **Create Audit** dialog box, which is displayed in Figure 3, opens, enabling you to configure the audit properties.

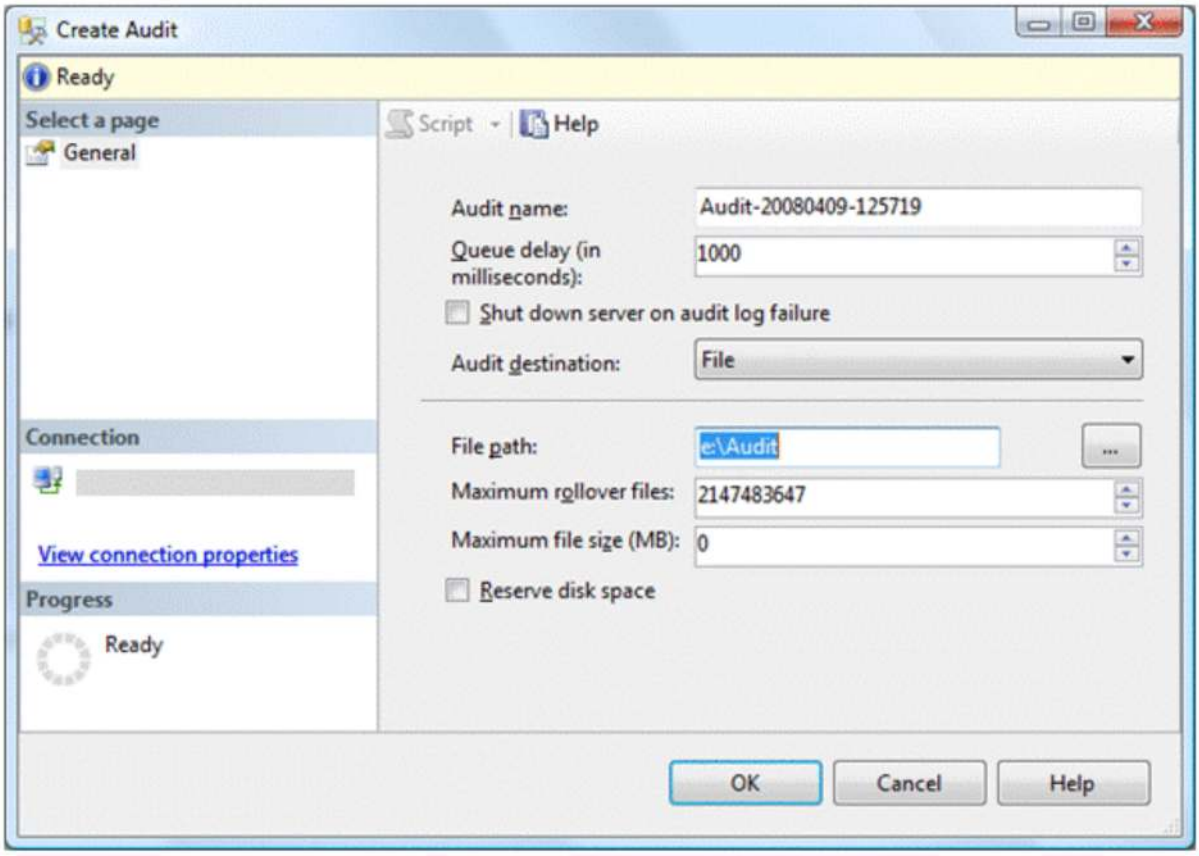

Figure 3: Create Audit dialog box

If the defaults are acceptable, the only input required is the **File path**, assuming that the audit destination is a file. Notice that SQL Server Management Studio created a default audit name based on the timestamp. Next the server and database audit specifications must be created. This can be accomplished by right-clicking on **Server Audit Specifications** and **Database Audit Specifications** in the Object Explorer respectively (see Figure 4).

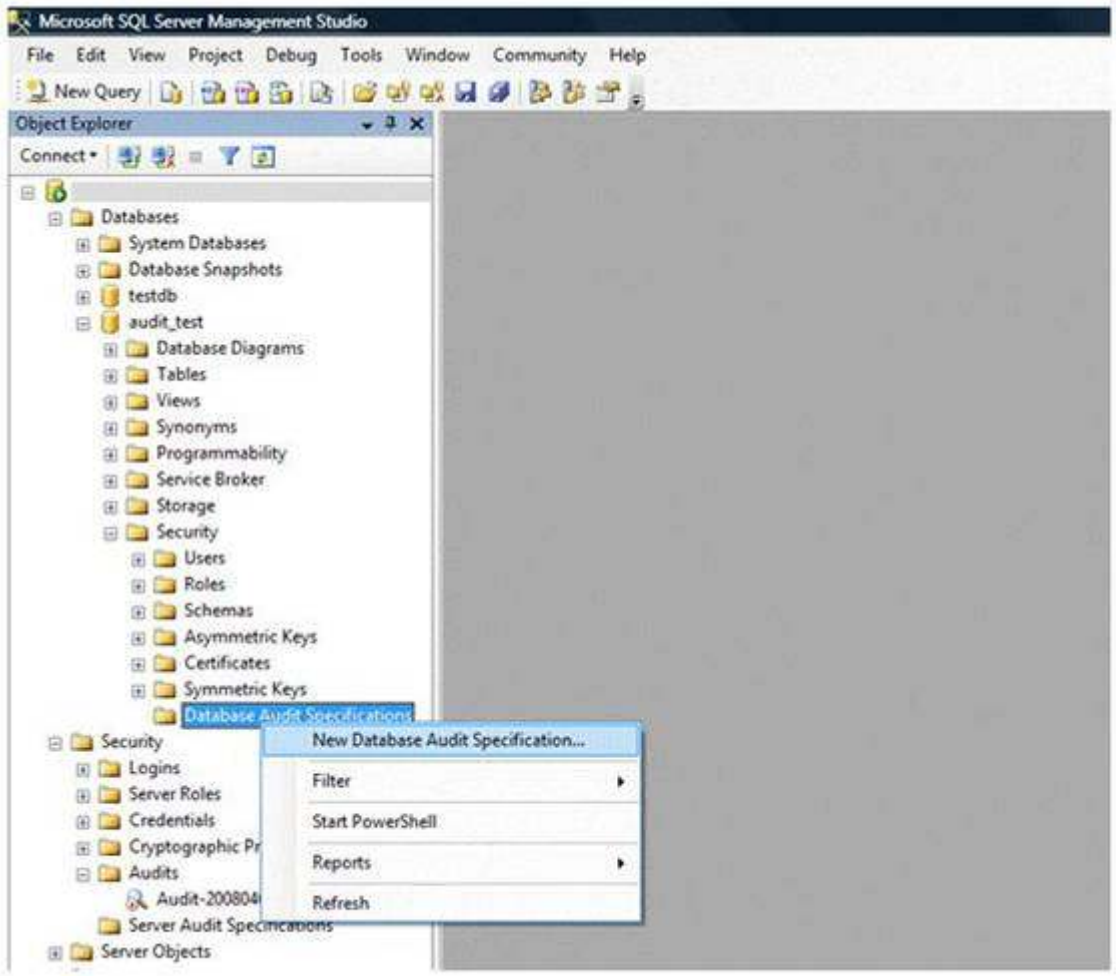

Figure 4: Creating a database audit specification through SQL Server Management Studio

For the database audit specification, the **Create Database Audit Specification** dialog box, which is shown in Figure 5, appears.

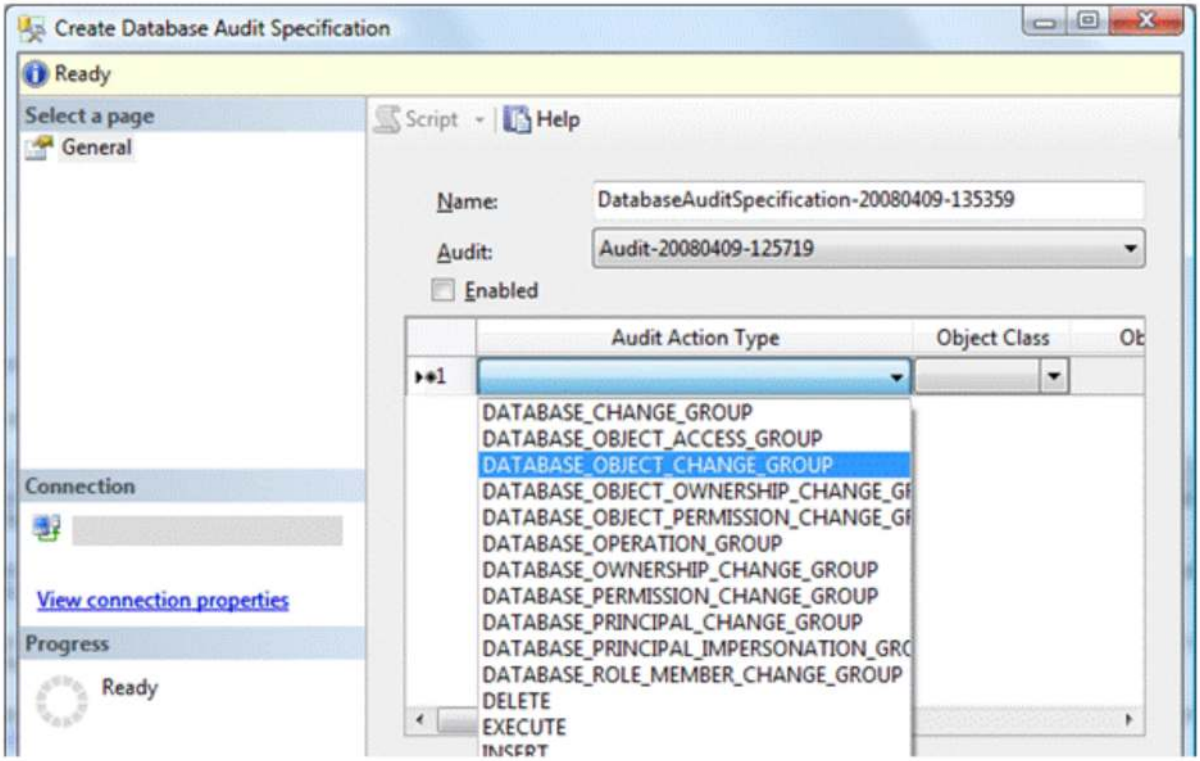

Figure 5: Create Database Audit Specification dialog box

The audit object created in the previous step needs to be selected in the **Audit** drop-down list. Next select DATABASE_OBJECT_CHANGE_GROUP from the **Audit Action Type** drop-down list.

The steps for creating a server audit specification are very similar. At this point, enable the audit and the related specifications by right-clicking them in the Object Explorer.

After the audit is running, administrators can leverage the abilities of Policy-Based Management (PBM) to manage the audits on multiple servers to see whether they are running and whether they are configured properly. A basic understanding of PBM is assumed in the subsequent discussion; for more information, see SQL Server Books Online.

Creating a policy to determine whether an audit or an audit specification is enabled is fairly straightforward. To create a policy that reports whether a server has an audit enabled, a new condition needs to be created using the audit facet and an expression that evaluates whether *@Enabled* equals True (see Figure 6).

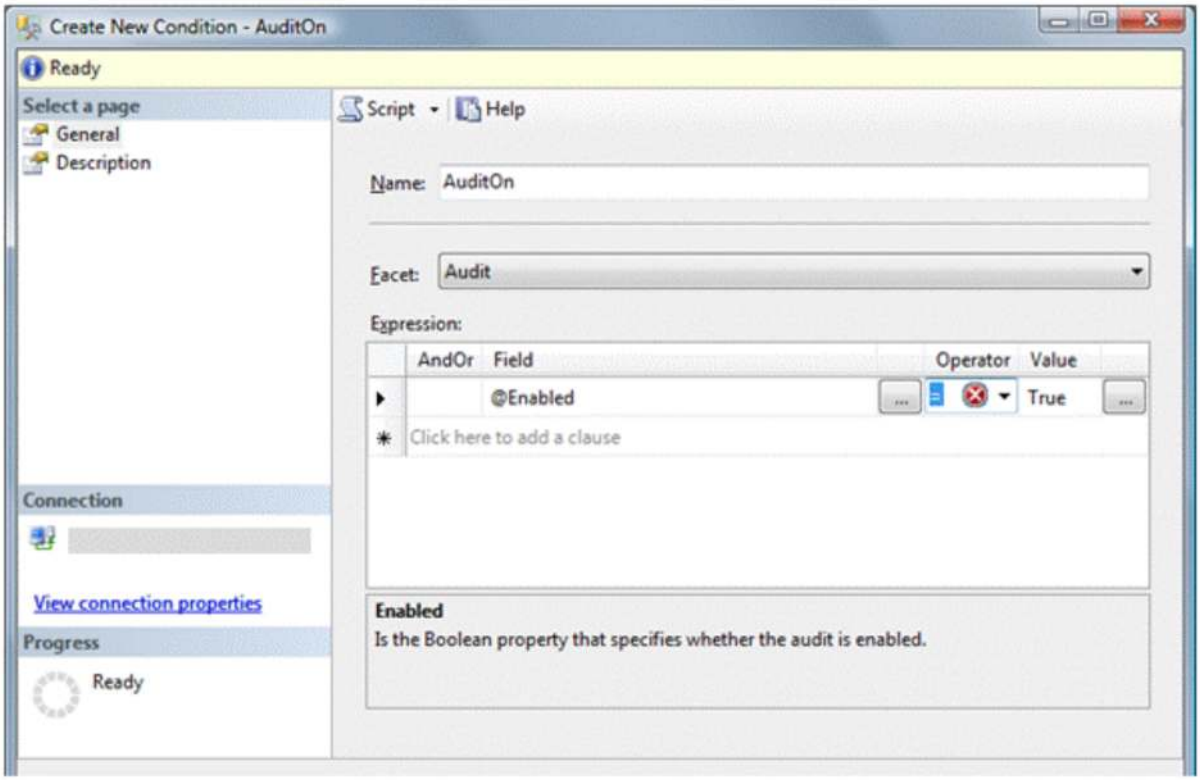

Figure 6: PBM Create New Condition dialog box

Note that the value in the **Name** box is arbitrary. Now that the condition is created, a new policy should be created. In this example, the new policy is given the name AuditOnPolicy, and the value in **Check Condition** is set to the AuditOn condition that was just created (Figure 7).

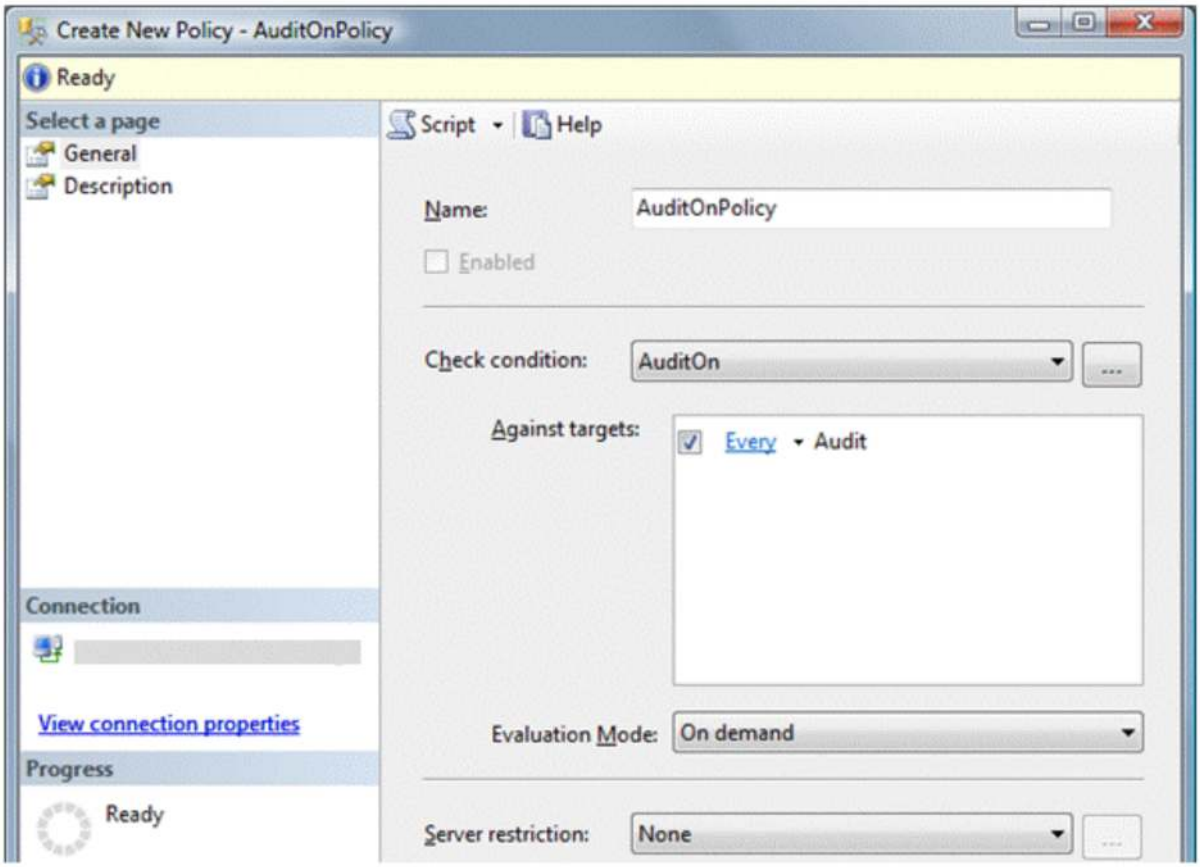

Figure 7: PBM Create New Policy dialog box

Clicking **OK** creates the policy that will evaluate whether the audit object on the server is enabled (or servers if the evaluation is initiated through the **Registered Servers** dialog box). Evaluation of the policy provides a list of audit objects defined in the instance and an indication of whether they are active or not. The results of the evaluation in Figure 8 indicate that one audit is not enabled.

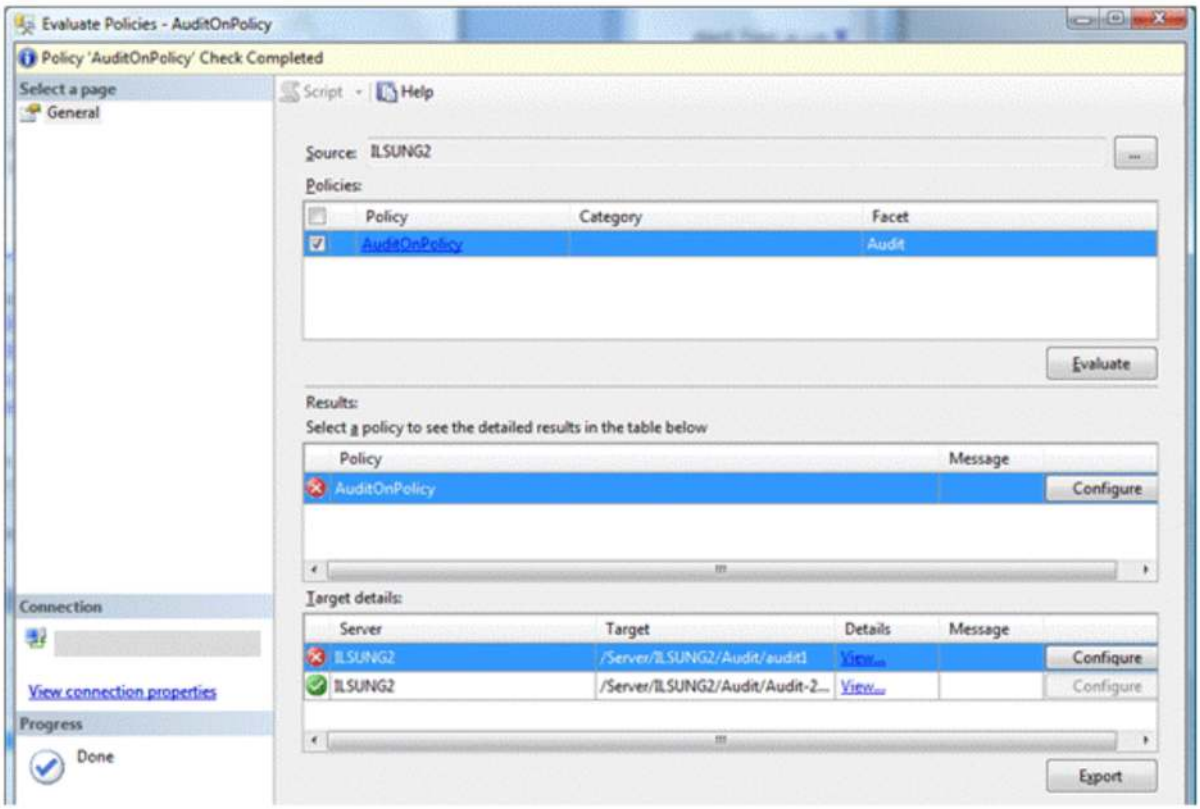

Figure 8: PBM Evaluate Policies dialog box

Please note that the audit objects are read-only in this release of SQL Server. Because of this, clicking **Configure** returns an error. Finally, a similar policy should be created to determine whether the audit specification(s) are enabled as well.

Of course, knowing that an audit is enabled is important but equally interesting is validating that the audit is configured properly. Although determining whether an action group, such as DATABASE_OBJECT_CHANGE_GROUP, has been defined in the audit specification might not be obvious, it is possible. Because no PBM facet exists for audit action groups, the policy can instead be described as a T-SQL query to determine whether a specific action group is included within the specification. One such query is provided here.

Transact-SQL Copy Code

```
SELECT 1
FROM sys.server_audits AS a
    JOIN sys.database_audit_specifications AS s
        ON a.audit_guid = s.audit_guid
    JOIN sys.database_audit_specification_details AS d
        ON s.database_specification_id = d.database_specification_id
WHERE a.is_state_enabled = 1
    AND s.is_state_enabled = 1
    AND d.audit_action_name = N'DATABASE_OBJECT_CHANGE_GROUP'
```

The query returns the value 1 if an active audit contains a specification created with the DATABASE_OBJECT_CHANGE_GROUP action group. To create a condition based on this query in PBM, do the following:

1. Create a new condition and in **Expression**, click on the "**...**" button to bring up the **Advanced Edit** dialog box.
2. Scroll down the **Functions and properties** list and double-click **ExecuteSql()**. This populates **Cell Value** with the text "ExecuteSql(string returnType, string sqlQuery)". You can edit this string to replace *string returnType* with *'numeric'* and *string sqlQuery* with the SELECT statement above (see Figure 9; note that

the single quote in the query must be escaped with another single quote).

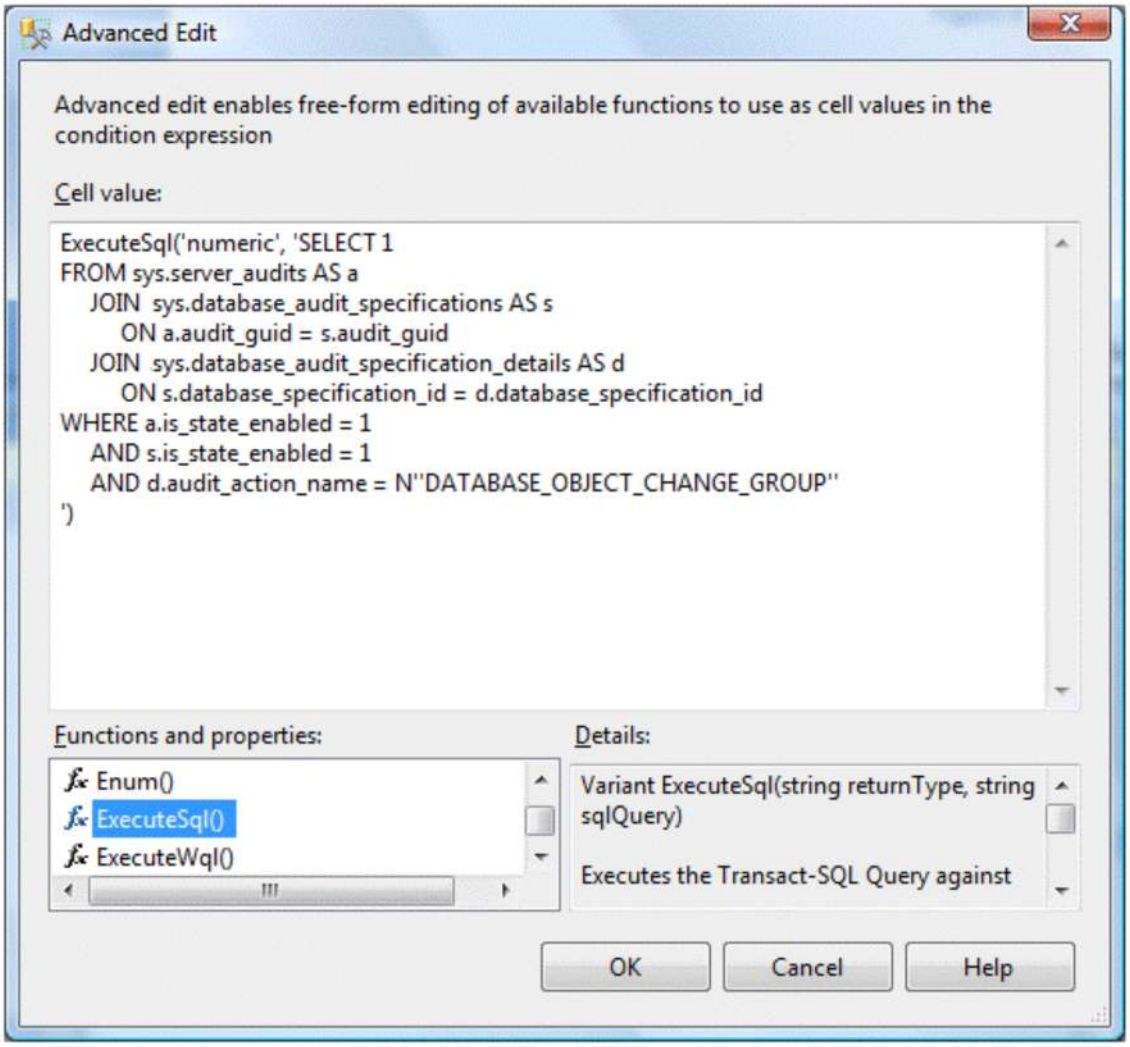

Figure 9: PBM Advanced Edit Window

3. After clicking **OK** and returning to the **Create New Expression** dialog box, complete the Expression by setting **Operator** to **=** and **Value** to **1**.

4. Set **Facet** to **Database** because you want this query evaluated for each database in the instance.

When evaluated, a policy using this expression will help the administrator detect when the database is not being audited properly.

Conclusion

The new SQL Server Audit feature introduced in SQL Server 2008 represents a significant improvement over the auditing capabilities offered in previous versions of SQL Server. One of the key advancements is the introduction of fine-grained auditing whereby events can be targeted to specific actions on an object by particular principals; the objects can even be scoped down to the individual table level. Performance is another big incentive, because SQL Server Audit performs significantly better than a comparable SQL Trace, up to around 45% in some cases. Additionally, the audit target is no longer confined to just files, because the Windows Application and Security logs are now options. And considering benefits such as persistence of the audit state between instance restarts and the involuntary recording of changes to the audit state, the new SQL Server Audit feature provides a robust and comprehensive auditing solution for the enterprise.

For more information:

http://msdn2.microsoft.com/en-us/library/cc280386(SQL.100).aspx [http://msdn2.microsoft.com/en-us/library/cc280386(SQL.100).aspx] : Understanding SQL Server Audit

http://msdn2.microsoft.com/en-us/library/bb630282(SQL.100).aspx [http://msdn2.microsoft.com/en-us/library/bb630282(SQL.100).aspx] : SQL Server Extended Events:

http://msdn2.microsoft.com/en-us/library/bb510667(SQL.100).aspx [http://msdn2.microsoft.com/en-us/library/bb510667(SQL.100).aspx] : Administering Servers by Using Policy-Based Management

http://www.microsoft.com/sqlserver/2008/en/us/security.aspx [http://www.microsoft.com/sqlserver/2008/en/us/security.aspx] : SQL Server 2008 Security

http://www.microsoft.com/sqlserver/ [http://www.microsoft.com/sqlserver/default.aspx] : SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/ [http://technet.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/ [http://msdn.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server DevCenter

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback [mailto://microsoft.com:25/default.aspx?subject=White%20Paper%20Feedback:%20Auditing%20in%20SQL%20Server%202008] .