Module 4: Protecting Data with Encryption and Auditing

Contents:

Module Overview

When configuring security for your Microsoft® SQL Server® systems, you should ensure that you meet any of your organization's compliance requirements for data protection. Organizations often need to adhere to industry-specific compliance policies, which mandate auditing of all data access. To address this requirement, SQL Server provides a range of options for implementing auditing.

Another common compliance requirement is the encryption of data to protect against unauthorized access in the event that access to the database files is compromised. SQL Server supports this requirement by providing transparent data encryption (TDE). To reduce the risk of information leakage by users with administrative access to a database, columns containing sensitive data—such as credit card numbers or national identity numbers—can be encrypted using the Always Encrypted feature.

This module describes the available options for auditing in SQL Server, how to use and manage the SQL Server Audit feature, and how to implement encryption.

Objectives

After completing this module, you will be able to:

- Describe the options for auditing data access.
- Implement SQL Server Audit.
- Manage SQL Server Audit.
- Describe and implement methods of encrypting data in SQL Server.

Lesson 1: Options for Auditing Data Access in SQL Server

SQL Server provides a variety of tools that you can use to audit data access. In general, no one tool meets all possible auditing requirements and a combination of features is often required.

In this lesson, you will learn about the different auditing options that are available.

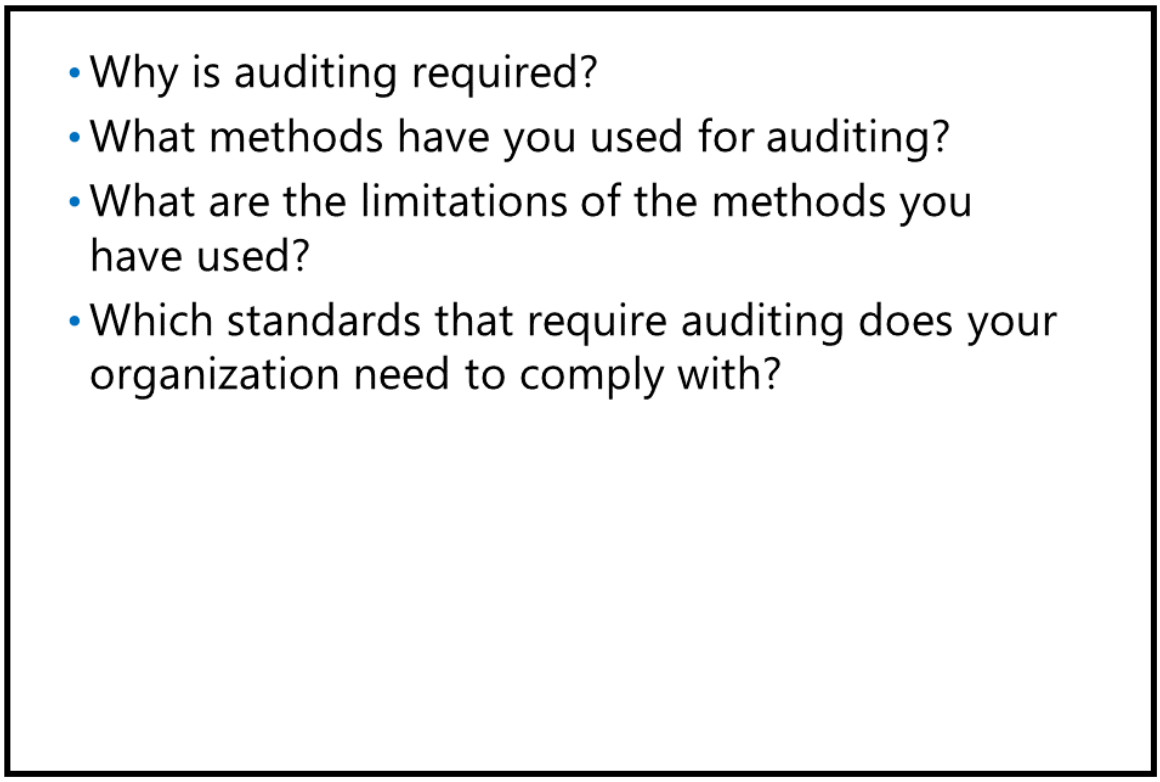
Lesson Objectives

At the end of this lesson, you will be able to:

- Describe why organizations might audit data access.
- Describe the Common Criteria compliance feature.
- Use triggers for auditing.
- Use temporal tables for auditing.

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Discussion: Auditing Data Access

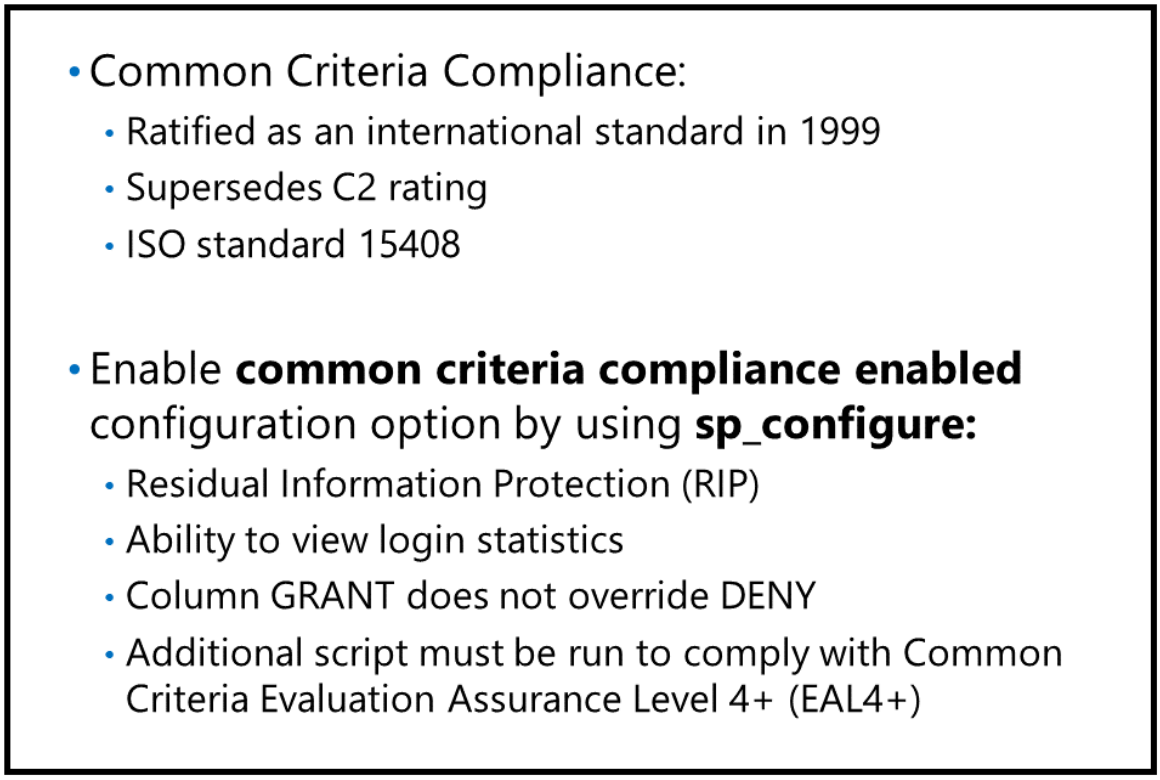


During this discussion, you will consider the following questions:

- Why is auditing required?
- What methods have you used for auditing?
- What are the limitations of the methods you have used?
- Which standards that require auditing does your organization need to comply with?

Enabling Common Criteria Compliance

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



Common Criteria is an international standard for computer security certification that was ratified by more than 20 nations in 1999, and has superseded the US C2 rating as a requirement in most standards. It is now maintained by more than 20 countries and is adopted by the International Standards Organization as ISO 15408.

> **Note:** SQL Server 2016 and previous versions also support the C2 audit mode; however, you should update any applications using C2 audit mode to use the Common Criteria certification.

> **Note:** Azure® SQL Database cannot currently be configured to comply with the Common Criteria.

Enabling Common Criteria Compliance in SQL Server

SQL Server provides the **common criteria compliance enabled** option, which can be set by using the **sp_configure** system stored procedure. The option is available in the Enterprise edition for use in production systems (it is also available in the Developer and Evaluation editions for nonproduction use).

When the option is enabled, three changes occur to how SQL Server operates:

- **Residual Information Protection (RIP)**. Memory is always overwritten with a known bit pattern before being reused.
- **Ability to view login statistics**. Auditing of logins is automatically enabled.
- **Column GRANT does not override table DENY**. This changes the default behavior of the permission system.

> **Note:** The implementation of RIP increases security, but can negatively impact the performance of the system.

To comply with Common Criteria Evaluation Assurance Level 4+ (EAL4+), in addition to enabling the **common criteria compliance enabled** option, you must also download and run a script that makes further configuration

changes to SQL Server. You can download this script from the Microsoft SQL Server Common Compliance website.

For more information on SQL Server's compliance with the Common Criteria, see *An introduction to the Common Criteria*:

An introduction to the Common Criteria

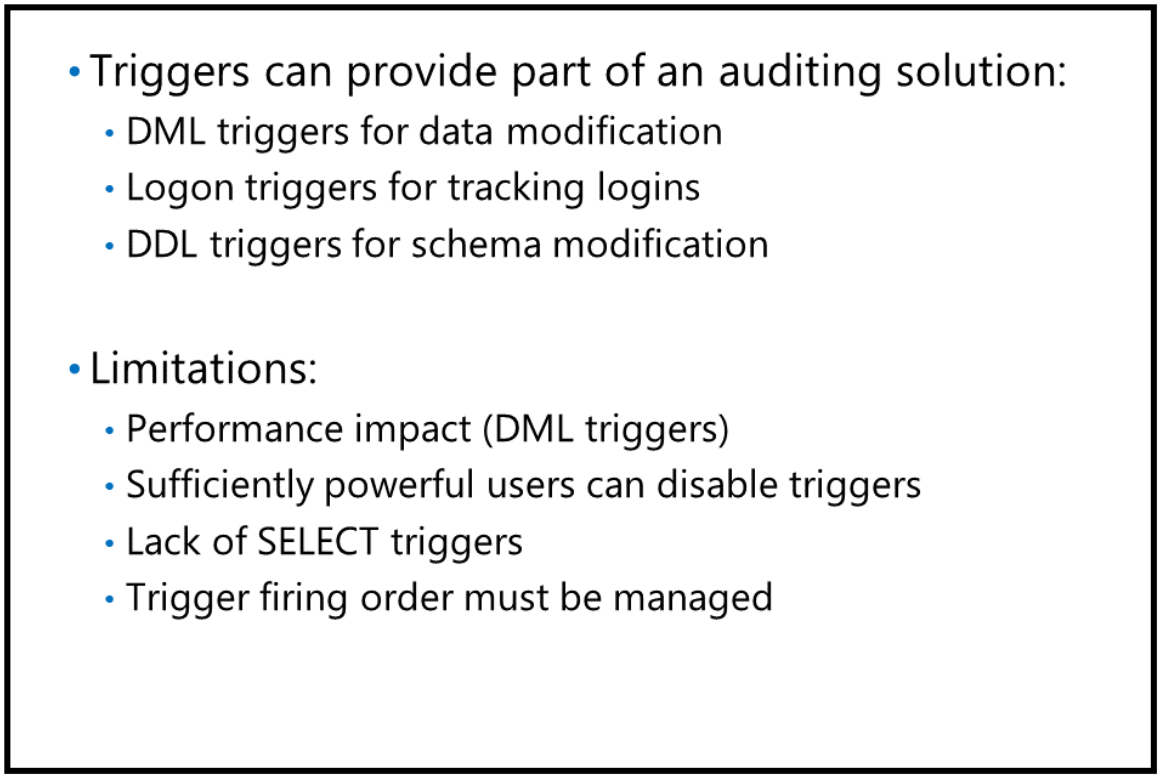
https://aka.ms/E6gtxr

For more information on the **common criteria compliance enabled** server option, see *common criteria compliance enabled Server Configuration Option* in Microsoft Docs:

common criteria compliance enabled Server Configuration Option

http://aka.ms/wifc4c

Auditing with Triggers



Triggers are a form of stored procedure that is triggered automatically in response to an event. SQL Server supports a variety of trigger types, including:

- **Data manipulation language (DML) triggers**. These triggers are associated with a table, and run when data in the table is modified.
- **Logon triggers**. A logon trigger runs in response to a login event.
- **Data definition language (DDL) triggers**. These triggers are associated with DDL statements that create, alter, or drop database objects.

> **Note:** Azure SQL Database includes support for DML triggers and DDL triggers. Logon triggers are not supported in Azure SQL Database.

All of these trigger types can play a role in auditing. All trigger types are created using the CREATE TRIGGER statement. In an auditing context, AFTER triggers are most often used.

DML Triggers

DML triggers are configured to fire when INSERT, UPDATE, and/or DELETE statements run against a table. You can access the original and new information by using the internal **inserted** and **deleted** tables in the trigger code. When used for auditing, triggers are commonly used to write details of a change to another table—the table holding the audited copy of the data might be in another database on the same SQL Server instance.

The following example shows a DML trigger that runs when an update occurs on a row in the **dbo.Employee** table. The original data is logged in the **dbo.EmployeeSalaryAudit** table.

A DML Auditing Trigger

```
CREATE TRIGGER TR_Employee_Salary_UPDATE
ON dbo.Employee
FOR UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  IF UPDATE(Salary) BEGIN
    INSERT dbo.EmployeeSalaryAudit (EmployeeID, OldSalary, NewSalary, UpdatedBy, UpdatedAt)
    SELECT i.EmployeeID, d.Salary, i.Salary, SUSER_NAME(), GETDATE()
    FROM inserted AS i
    JOIN deleted AS d
    ON i.EmployeeID = d.EmployeeID;
  END;
END;
GO
```

For more information on DML triggers, see *DML Triggers* in Microsoft Docs:

DML Triggers

http://aka.ms/ys2yfy

Logon Triggers

A logon trigger fires in response to a login that is authenticated but before a session is established. Logon triggers can be used to record the logon, either to a table or to the SQL Server error log.

For more information on logon triggers, see *Logon Triggers* in Microsoft Docs:

Logon Triggers

http://aka.ms/lqyc2g

DDL Triggers

A DDL trigger fires in response to a DDL event that creates, drops, and/or alters either a specific class of database object or all database objects. In an auditing context, DDL triggers are commonly used to track changes to the schema of a database by recording DDL statements to a table.

For more information on DDL triggers, see *DDL Triggers* in Microsoft Docs:

DDL Triggers

http://aka.ms/qzkssj

Limitations

Triggers have some limitations as an audit tool:

- System performance can be significantly impacted by DML triggers running alongside the usual load on the server.
- Users with appropriate permissions can disable triggers. This can be a significant issue for auditing requirements.
- DML triggers cannot be used to audit data access through SELECT statements.
- Only limited ability to control trigger firing order is provided. To make sure that it captures all the changes made by other triggers, auditing would normally need to be the last DML trigger that fires—which you can only specify by using the **sp_settriggerorder** system procedure.

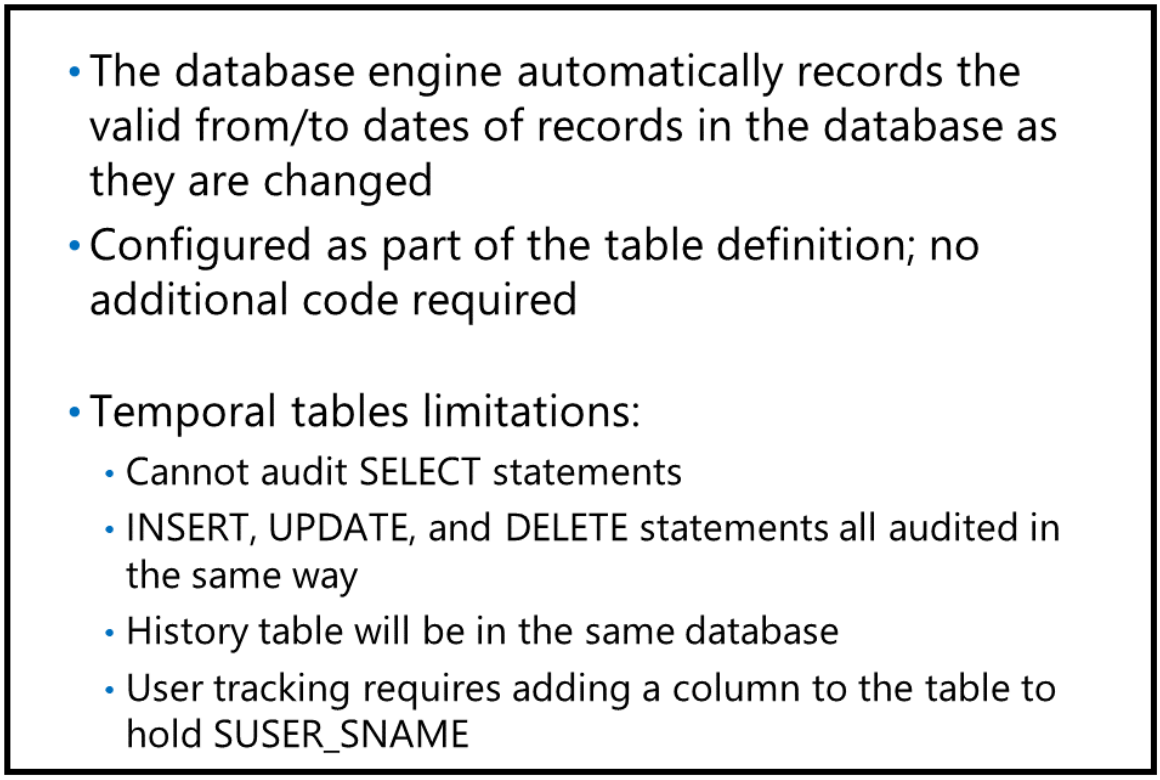
For more information on triggers, see *CREATE TRIGGER (Transact-SQL)* in Microsoft Docs:

CREATE TRIGGER (Transact-SQL)

http://aka.ms/nh99i4

Auditing with Temporal Tables

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



System-versioned temporal tables provide a mechanism for capturing data changes to a table into a history table; in most respects, this is similar to the method for auditing data changes with DML triggers that you learned about in the previous topic, except that the database engine manages the history table automatically.

You can use the temporal table feature to record all changes to your data. To use the feature, you start by creating a system-versioned table, either by creating a new table or modifying an existing one. When you create the new table, SQL Server actually creates two tables—one to store the current data and one to store historical data. Two **datetime2** columns are added to both the current and historical tables where SQL Server stores the valid date range of the data: **SysStartTime** and **SysEndTime**. The current row will have a **SysEndTime** value of 9999-12-31, and all records inserted within a single transaction will have the same UTC time. When a user updates a row, SQL Server copies the data to the historical table before the update; it also sets the **SysEndTime** column to the date when the data is changed. It then updates the row in the current table to reflect the change.

Create a new table and set the SYSTEM_VERSIONING feature ON.

Create Temporal Table

```
CREATE TABLE dbo.Employee (
              EmployeeID int NOT NULL PRIMARY KEY CLUSTERED,
              ManagerID int NULL,
              FirstName varchar(50) NOT NULL,
              LastName varchar(50) NOT NULL,
              SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL,
              SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL,
              PERIOD FOR SYSTEM_TIME (SysStartTime,SysEndTime)
) WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory));
GO
```

When used as an auditing mechanism, temporal tables have some limitations:

- Temporal tables cannot be used to audit data access through SELECT statements.

- When auditing with temporal tables, you cannot define different actions for INSERT, UPDATE, and DELETE statements—unlike DML triggers.
- The history table used to create a temporal table must be created in the same database as the current table.
- Tracking the identity of the user who made a change requires that you alter the definition of the table to include a column that defaults to SUSER_SNAME.

For more information on working with temporal tables, see *Temporal Tables* in Microsoft Docs:

Temporal Tables

http://aka.ms/ft8fc2

Demonstration: Auditing with Temporal Tables

In this demonstration, you will see a method for using temporal tables as an audit tool.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and **20764C-MIA-SQL** virtual machines are running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. **Start SQL Server Management Studio and connect to your Azure instance running the AdventureWorksLT database, using SQL Server authentication. In the Login box, type Student, in the Password box, type Pa55w.rd, and then click Connect.**
3. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod04\Demo** folder.
4. Open the **Demo 01 - temporal table audit.sql** query, and connect to the **AdventureWorksLT** database.
5. Execute the code under the heading for **Step 2** to create a system-versioned temporary table.
6. Execute the code under the heading for **Step 3** to insert some example data.
7. Execute the code under the heading for **Step 4** to update a row.
8. Execute the code under the heading for **Step 5** to examine the current and history tables that make up the temporal table.
9. Execute the code under the heading for **Step 6** to demonstrate the behavior of the FOR SYSTEM TIME ALL subclause.
10. Execute the code under the heading for **Step 7** to demonstrate the behavior of the FOR SYSTEM TIME AS OF subclause.
11. Execute the code under the heading for **Step 8** to demonstrate that the history table cannot be edited. Both commands will generate an error.
12. Execute the code under the heading for **Step 9** to demonstrate that a user with permission to update the table directly can insert misleading information.
13. Execute the code under the heading for **Step 10** to examine the temporal table again after the update.
14. When you have finished the demonstration, execute the code under the heading for **Step 11** to remove the demonstration objects.
15. Close SQL Server Management Studio, without saving any changes.

Check Your Knowledge

Categorize Activity

Categorize each audit method into the appropriate category. Indicate your answer by writing the category number to the right of each item.

Temporal Tables

Triggers

SQL Server Profiler

Records Changes to Data or Database Objects

Please drag items here

Records DML Statements or DDL Statements

Please drag items here

Check answer Show solution Reset

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Lesson 2: Implementing SQL Server Audit

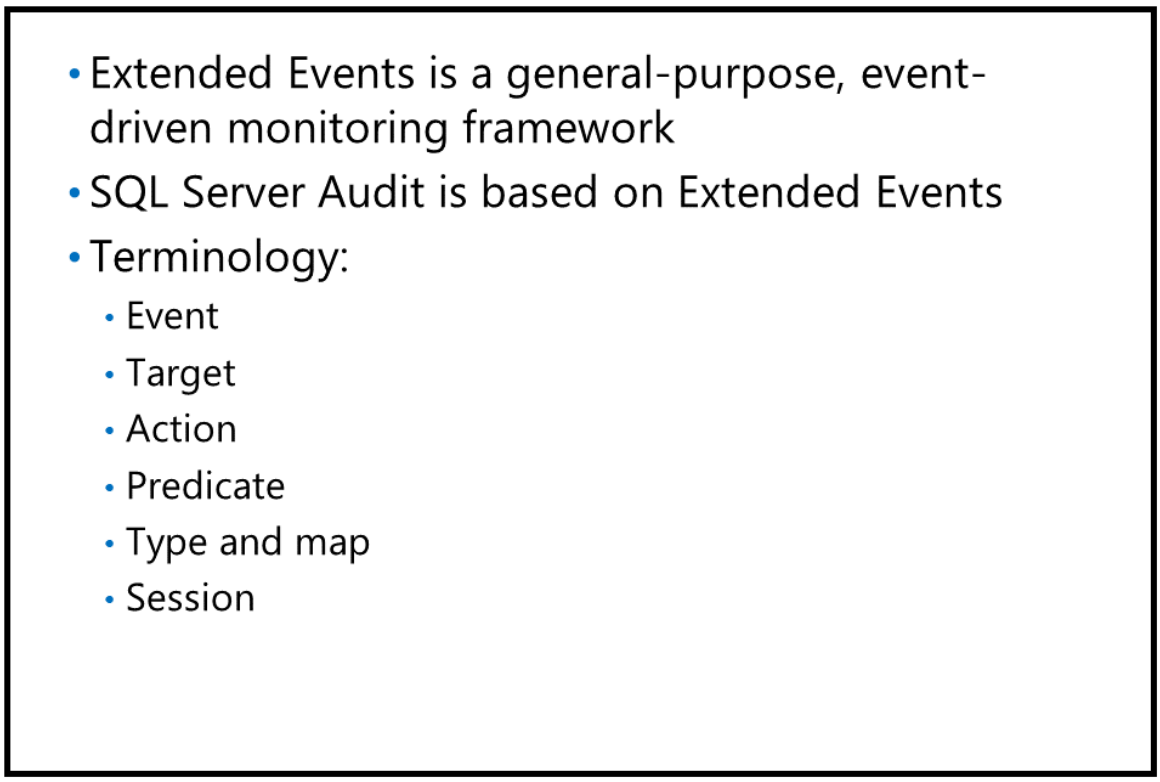
SQL Server includes a purpose-built auditing tool—SQL Server Audit. This lesson covers the architecture of SQL Server Audit, and how to configure and work with it.

Lesson Objectives

At the end of this lesson, you will be able to:

- Describe Extended Events.
- Describe SQL Server Audit.
- Create audits.
- Explain audit actions and action groups.
- Create server audit specifications.
- Create database audit specifications.
- Monitor audits using audit-related dynamic management objects and system views.
- Use custom audit events.
- Enable auditing in SQL Azure Database.

Introduction to Extended Events



SQL Server Audit is based on an event-driven monitoring engine called Extended Events.

Extended Events is an event-driven activity monitoring tool which follows a loose-coupled design pattern. Events and their targets are not tightly coupled; any event can be bound to any target. This means that data processing and filtering can be carried out independently of data capture, reducing the performance overhead of other auditing solutions.

Extended Events allows sophisticated filters to be defined on captured data. In addition to value filters, events may be filtered by sampling. Data may be aggregated at the point it is captured. Extended Events can be managed either through a GUI in SQL Server Management Studio or through Transact-SQL statements.

Extended Events can be integrated with the Event Tracing for Windows (ETW) framework, meaning SQL Server activity can be monitored alongside other Windows® components.

Extended Events is important because SQL Server Audit is based on the Extended Events infrastructure. The Extended Events engine is not tied to particular types of events—the engine is written in such a way that it can process any type of event.

> **Additional Reading:** Because Extended Events is the basis for SQL Server Audit, you could opt to write your own auditing system based on Extended Events. See Module 12 of this course *Tracing Access to SQL Server with Extended Events* for more information on working with Extended Events.

Executables and executable modules can expose one or more Extended Events packages at run time. Packages act as containers for the Extended Events objects and their definitions; a package may expose any of the following object types:

- **Events**. Points of interest reached during code execution.
- **Targets**. Logging targets, such as files in the file system.
- **Actions**. Additional data that can be collected when an event is triggered.

- **Predicates**. Filters used to restrict which events are captured.
- **Types** and **Maps**. Reference data; data type and lookup value definitions.
- **Sessions**. Links events and actions, filtered by predicates, to one or more targets. Typically, sessions are user-defined.

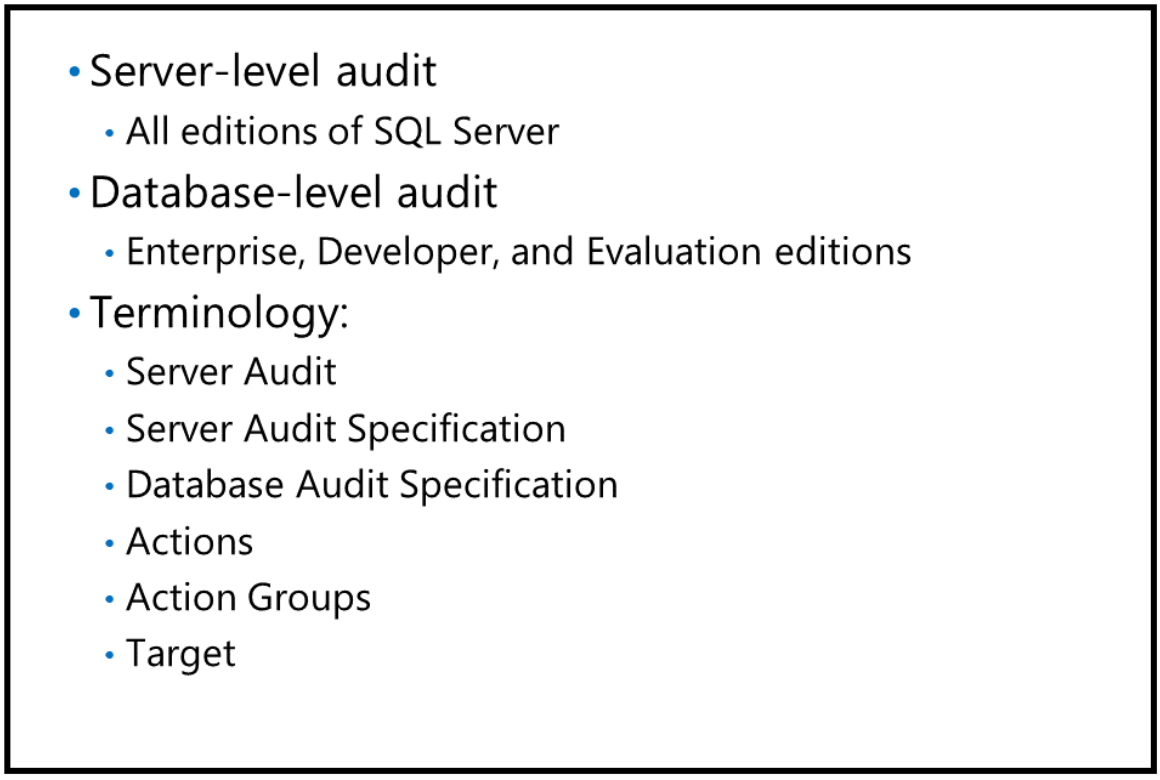
SQL Server Audit is a special package within Extended Events; you cannot change its internal configuration.

For more information on Extended Events, see *Extended Events* in Microsoft Docs:

Extended Events

http://aka.ms/b8p2e9

Introduction to SQL Server Audit



SQL Server Audit is the primary auditing tool in SQL Server. You can use it to track server-level and database-level events on an instance of SQL Server, and then log these events to audit files or event logs. All editions of SQL Server support server-level auditing. Enterprise edition, Developer edition, and Evaluation edition also support database-level auditing.

Because it is based on Extended Events, the SQL Server Audit terminology is similar to Extended Events terminology:

Object	Description
Server Audit	Defines where to store the audit data.
Server Audit Specification	Collects many server-level action groups raised by Extended Events. One per audit.
Database Audit Specification	Collects database-level audit actions raised by Extended Events. One per database per audit.
Actions	Specific actions that can raise events and be added to the audit. For example, SELECT operations on a table.

Object	Description
Action Groups	Logical groups of actions to simplify creating audit specifications. For example, BACKUP_RESTORE_GROUP, which includes any backup or restore commands.
Target	Receives and stores the results of the audit. Can be a file, Windows Security event log, or Windows Application event log.

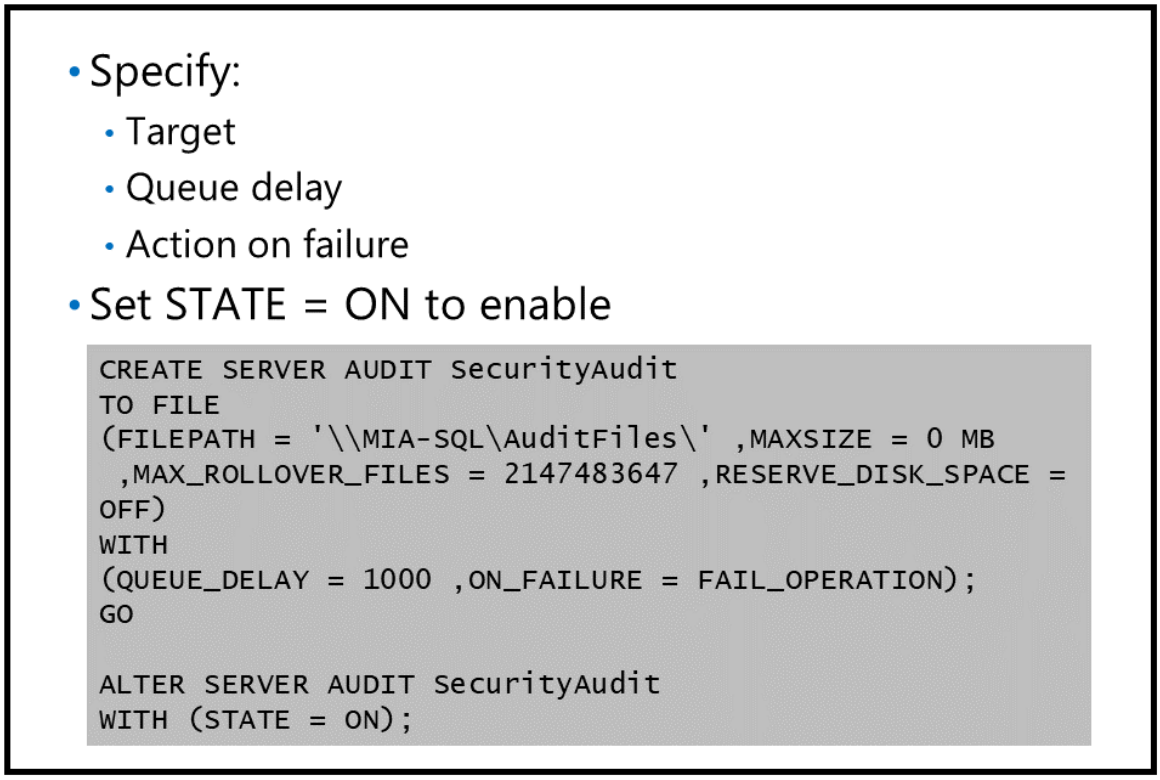
For more information about SQL Server Audit, see *SQL Server Audit (Database Engine)* in Microsoft Docs:

SQL Server Audit (Database Engine)

http://aka.ms/ucupsq

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Defining a Server Audit



A server audit defines where and how audited events are logged. Each server audit defines a target (such as a file or a Windows event log), the time interval before events must be written, and the action SQL Server should take if the target runs out of disk space.

You can create audits by using the CREATE SERVER AUDIT statement or through the SQL Server Management Studio (SSMS) GUI. There are several options you can configure, including:

- **Name**. User-friendly name to refer to the audit.
- **Queue delay**. Time in milliseconds that SQL Server can buffer the audit results before flushing them to the target.
- **On failure**. Action to take if the audit log is unavailable—continue, shut down server, or fail auditable operations.
- **Audit destination**. Location of the target.
- **Maximum file size**. Maximum size of each audit file (in MB).
- **Reserve disk space**. Indicates whether to reserve disk space for audit files in advance.

> **Note:** The value you configure for the queue delay needs to be a tradeoff between security and performance. A low value ensures that events are logged quickly and avoids the risk of losing items from the audit trail in the event of failure, but can result in a significant performance overhead.

When you create an audit, it will be in a disabled state.

Audit Targets

Audits can be sent to one of the following three targets:

- **Binary file**. File output provides the highest performance and is the easiest option to configure.
- **Windows Application Event Log**. Avoid sending too much detail to this log as network administrators tend to dislike applications that write too much content to any of the event logs. Do not use this target for sensitive data because any authenticated user can view the log.
- **Windows Security Event Log**. This is the most secure option for auditing **data, but you need to add the SQL Server service account to the Generate Security Audits policy before using it.**

You should review the contents of the target that you use and archive its contents periodically.

The following code example creates and enables a server audit that uses a binary file as the target:

Creating a Server Audit

```
CREATE SERVER AUDIT HR_Audit
    TO FILE (FILEPATH='\\MIA-SQL\Audit\')
    WITH (QUEUE_DELAY = 1000);
GO

ALTER SERVER AUDIT HR_Audit WITH (STATE = ON);
GO
```

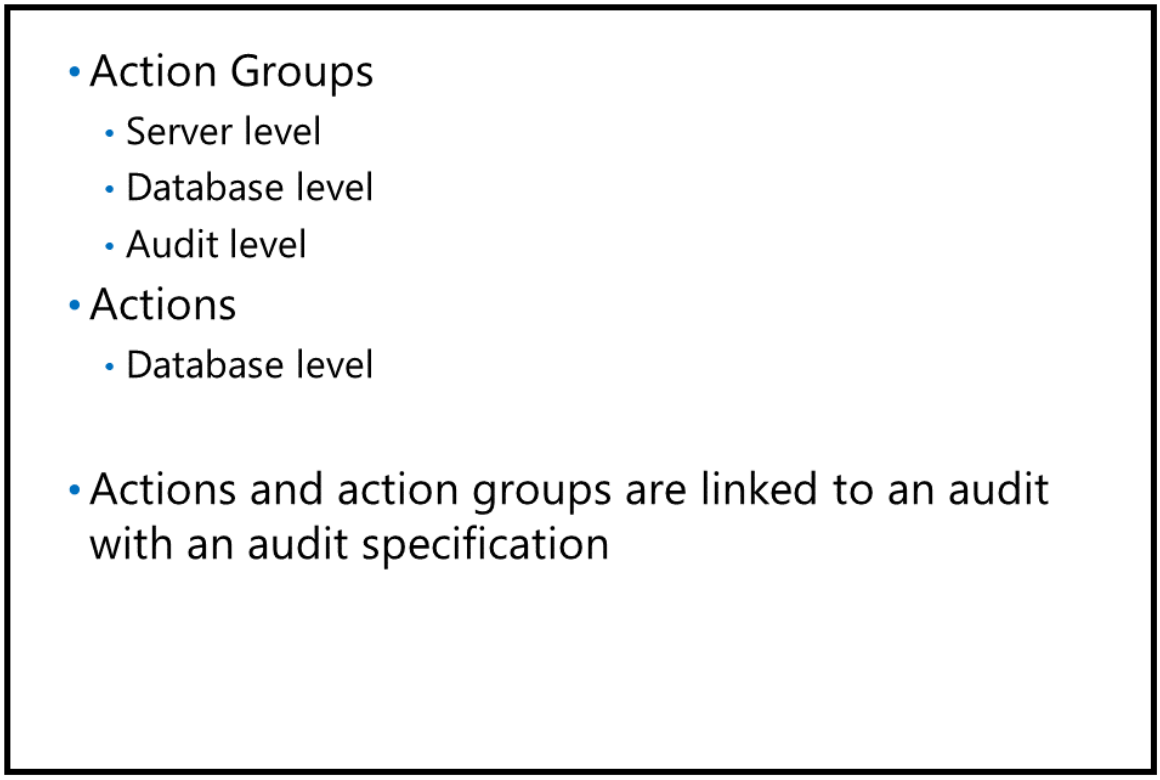
> **Note:** The filename that you provide to the FILEPATH parameter when creating a server audit is actually a path to a folder. SQL Server generates log files automatically and stores them in this location.

For more information on the CREATE SERVER AUDIT command, see *CREATE SERVER AUDIT (Transact-SQL)* in Microsoft Docs:

CREATE SERVER AUDIT (Transact-SQL)

http://aka.ms/mn06tw

Audit Actions and Action Groups



After you have defined an audit, you can specify the events that you want the audit to track; these events may be at server level, database level, or audit level.

> **Note:** Audit level events are triggered when an audit object is added, removed, or changed. Audit level events may be tracked at server level or database level.

You may add events to be tracked by an audit at the level of individual events, referred to as actions—such as a SELECT statement on a specific table—or in predefined action groups that collect related actions together—such as SUCCESSFUL_LOGIN_GROUP, which includes all the actions connected to a successful login.

Actions and action groups are linked to an audit through an audit specification.

Predefined action groups are available at server level, database level, and audit level. Actions are only available at database level.

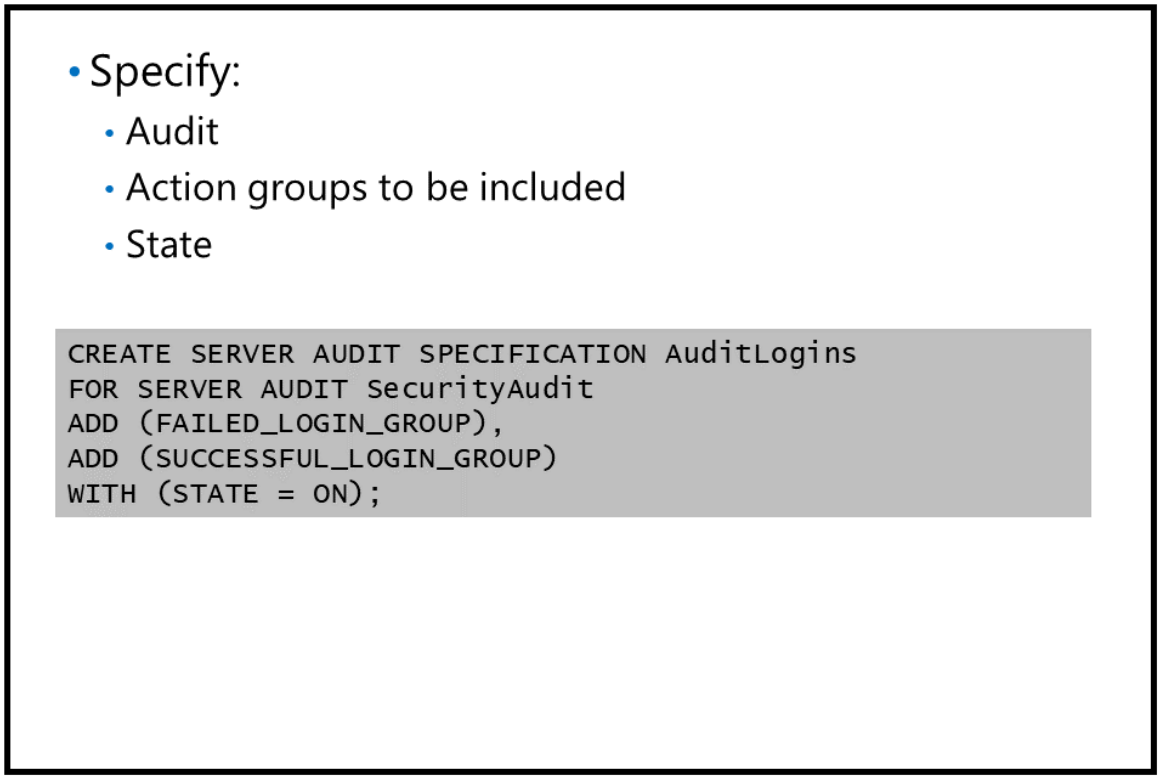
For a full list of available server actions and action groups, see *SQL Server Audit Action Groups and Actions* in Microsoft Docs:

SQL Server Audit Action Groups and Actions

http://aka.ms/bak8rw

Creating Server Audit Specifications

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



After you create an audit, you can add server audit specifications to include one or more action groups in the audit. When you create a server audit specification, it is automatically disabled, so you must remember to enable it when you are ready for it to start.

A server audit specification details the actions to audit. Server-level actions are grouped into predefined action groups, including:

- BACKUP_RESTORE_GROUP. Includes all backup and restore actions.
- DATABASE_CHANGE_GROUP. Includes actions that create, alter, or drop a database.
- FAILED_LOGIN_GROUP. Includes details of failed login attempts.
- SUCCESSFUL_LOGIN_GROUP. Includes details of successful logins.
- SERVER_OPERATION_GROUP. Includes actions that alter server settings.

For a full list of available server action groups, see *Server-Level Audit Action Groups* in *SQL Server Audit Action Groups and Actions* in Microsoft Docs:

SQL Server Audit Action Groups and Actions - Server-Level Audit Action Groups

http://aka.ms/bak8rw

The following example creates and enables a server audit specification to track failed and successful login attempts. This code assumes that the server audit **SecurityAudit** has already been created.

Creating a Server Audit Specification

```
CREATE SERVER AUDIT SPECIFICATION AuditLogins
FOR SERVER AUDIT SecurityAudit
ADD (FAILED_LOGIN_GROUP),
```

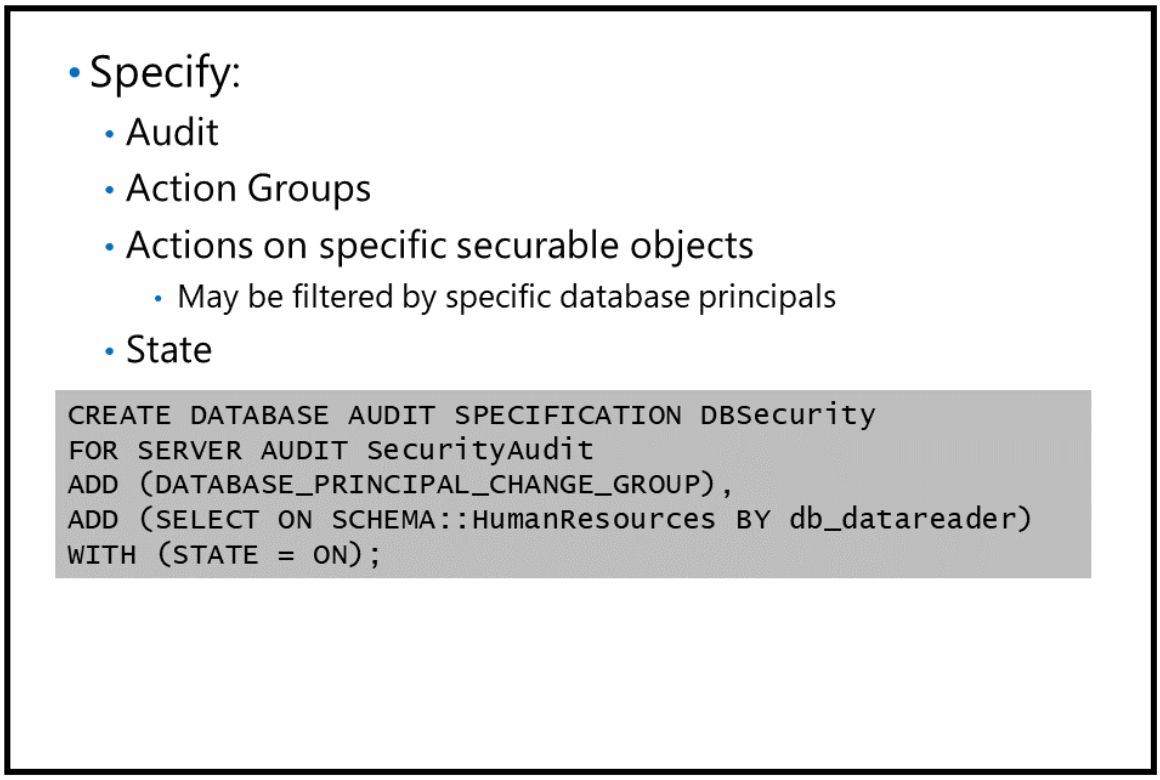
```
ADD (SUCCESSFUL_LOGIN_GROUP)
WITH (STATE = ON);
```

For more information on the CREATE SERVER AUDIT SPECIFICATION command, see *CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)* in Microsoft Docs:

CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)

http://aka.ms/rkn63h

Creating Database Audit Specifications



Creating a database audit specification is similar to creating a server audit specification. The database audit specification details actions or action groups to audit at the database level.

> **Note:** For production systems, database level auditing is only available in SQL Server Enterprise edition.

You can audit individual database-level actions, including:

- SELECT. Occurs when a SELECT statement is issued.
- INSERT. Occurs when an INSERT statement is issued.
- DELETE. Occurs when a DELETE statement is issued.
- UPDATE. Occurs when an UPDATE statement is issued.
- EXECUTE. Occurs when an EXECUTE statement is issued.

Additionally, SQL Server defines database-level action groups, including:

- APPLICATION_ROLE_CHANGE_PASSWORD_GROUP. Occurs when an application role password is changed.
- DATABASE_OBJECT_ACCESS_GROUP. Includes all access to database objects.
- DATABASE_PRINCIPAL_CHANGE_GROUP. Includes all events when a database-level principal is created, altered, or dropped.
- SCHEMA_OBJECT_ACCESS_GROUP. Includes all access to objects in a specified schema.
- SCHEMA_OBJECT_PERMISSION_CHANGE_GROUP. Includes all changes to object permissions.

For a full list of available database actions and action groups, see *Database-Level Audit Action Groups* and *Database-Level Audit Actions* in *SQL Server Audit Action Groups and Actions* in Microsoft Docs:

SQL Server Audit Action Groups and Actions - Database-Level Audit Action Groups and Database-Level Audit Actions

http://aka.ms/bak8rw

The following example creates an audit specification that includes all database principal changes and all SELECT queries on objects in the **salesapp1** schema by members of the **db_datareader** fixed database-level role. This code assumes that the server audit **SecurityAudit** has already been created.

Creating a Database Audit Specification

```
CREATE DATABASE AUDIT SPECIFICATION DBSecurity
FOR SERVER AUDIT SecurityAudit
ADD (DATABASE_PRINCIPAL_CHANGE_GROUP),
ADD (SELECT ON SCHEMA::salesapp1 BY db_datareader)
WITH (STATE = ON);
```

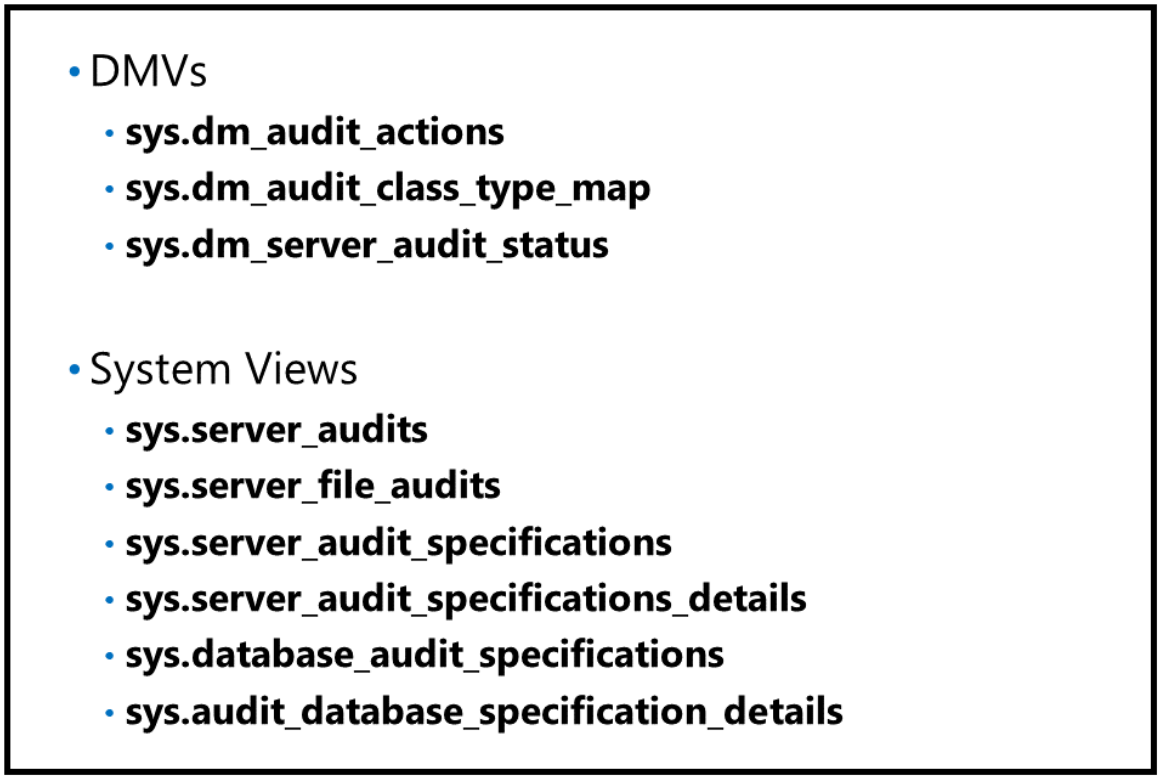
For more information on the CREATE DATABASE AUDIT SPECIFICATION command, see CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL) in Microsoft Docs:

CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL)

http://aka.ms/x6x4qu

Audit-Related Dynamic Management Views and System Views

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



SQL Server provides a number of dynamic management views (DMVs) and system views that can help you retrieve SQL Server Audit configuration information.

Audit DMVs

The following DMVs return metadata about audits:

- **sys.dm_audit_actions**. Returns a complete list of available audit actions and audit action groups.
- **sys.dm_audit_class_type_map**. Returns a reference data list mapping audit class codes to descriptions.
- **sys.dm_server_audit_status**. Returns a list of all the audits defined on an instance of SQL Server, and their current status.

For more information on audit-related DMVs, see the links in *Security-Related Dynamic Management Views and Functions (Transact-SQL)* in Microsoft Docs:

Security-Related Dynamic Management Views and Functions (Transact-SQL)

http://aka.ms/u0cafg

Audit System Views

The following system views return metadata about audits:

- **sys.server_audits**. Returns a list of all the audits defined on an instance of SQL Server.
- **sys.server_file_audits**. Returns a list of all audits that write data to a file target.
- **sys.server_audit_specifications**. Returns a list of high level information about server audit specifications on an instance.
- **sys.server_audit_specifications_details**. Returns a list of the action groups associated with each server audit

specification. This view can be joined to **sys.dm_audit_actions** to resolve action group names.

- **sys.database_audit_specifications**. Returns a list of high level information about database audit specifications on an instance.
- **sys.audit_database_specification_details**. Returns a list of the actions and action groups associated with each database audit specification. This view can be joined to **sys.dm_audit_actions** to resolve action and action group names.

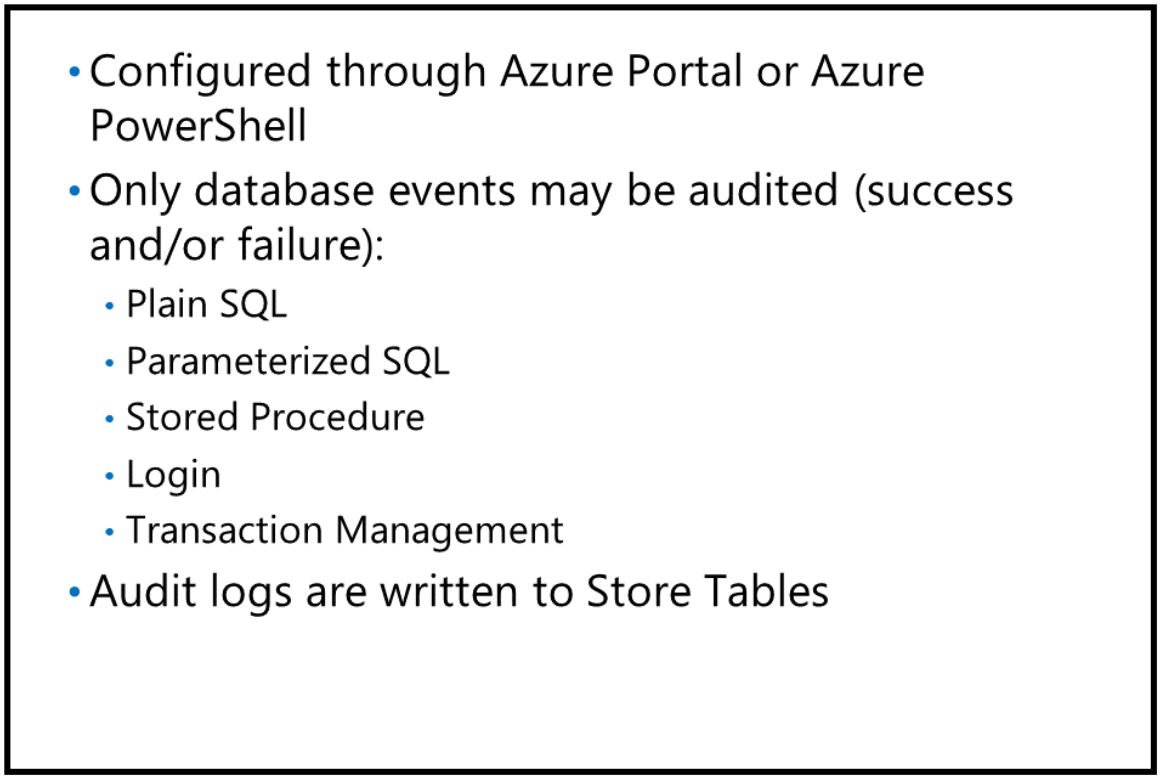
This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

For more information on audit-related system views, see the links in *SQL Server Audit Views* section of *Security Catalog Views (Transact-SQL)* in Microsoft Docs:

Security Catalog Views (Transact-SQL) - SQL Server Audit Views

http://aka.ms/h1asxy

Auditing in Azure SQL Database



Auditing in Azure SQL Database is less complex than for an installation of SQL Server, and is configured in a different way. Azure SQL Database auditing cannot be configured using Transact-SQL commands or through SSMS; instead, it is configured through the Azure portal, or with Azure PowerShell™.

Although auditing is only available on database-level actions, Azure SQL Database auditing can be configured either at the level of individual databases or at database server level—the server-level audit setting will be applied to all the databases hosted on the server.

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

You may enable auditing on the following event categories:

- Plain SQL
- Parameterized SQL
- Stored Procedure

- Login
- Transaction Management

You may choose to audit success, failure, or both success and failure of each event category.

Audit data is held in a collection of Store Tables in a storage account that you select.

Auditing in Azure SQL Database is available for all service tiers (Basic, Standard, and Premium).

For more information and instructions on how to configure auditing in Azure SQL Database, see *Get started with SQL database auditing* in the Azure documentation:

Get started with SQL database auditing

http://aka.ms/bnbvzb

Demonstration: Using SQL Server Audit

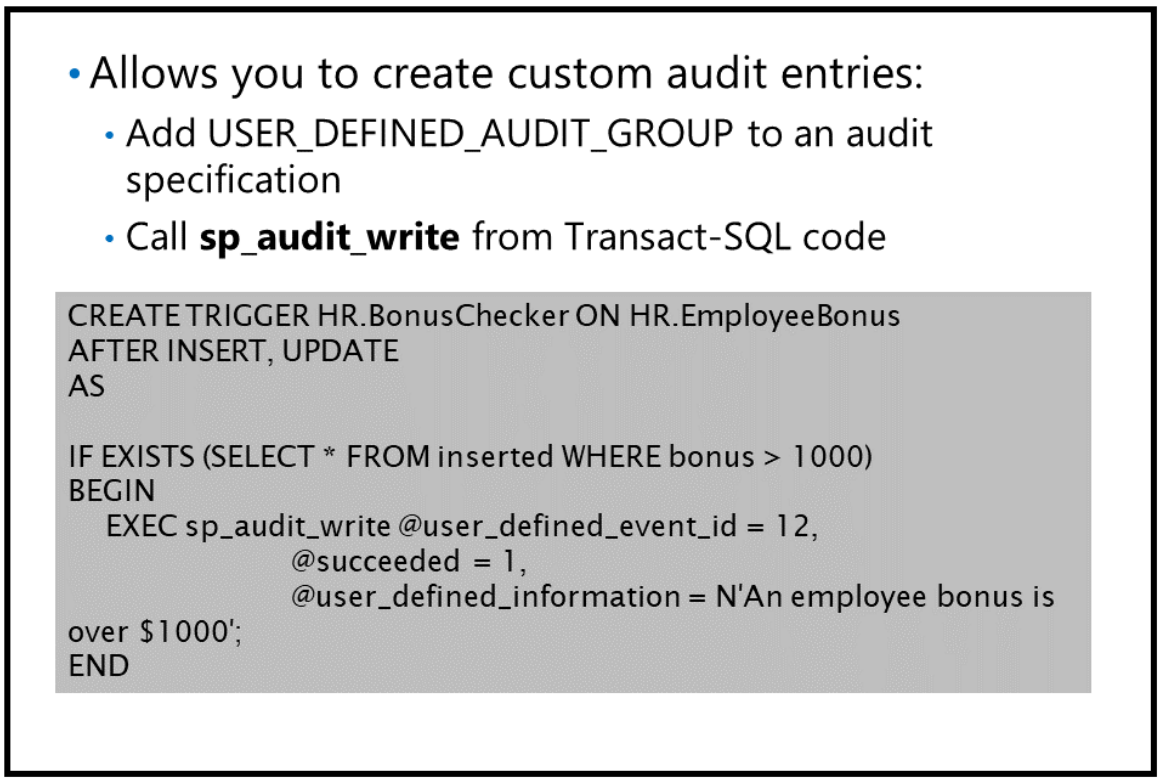
In this demonstration, you will see how to configure SQL Server Audit.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and **20764C-MIA-SQL** virtual machines are running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod04** folder as Administrator.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to complete.
4. Start SQL Server Management Studio and connect to **MIA-SQL** using Windows authentication.
5. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod04\Demo** folder.
6. Open the **Demo 02 - audit.sql** query.
7. Execute the code under the heading for **Step 1** to create a new audit.
8. Execute the code under the heading for **Step 2** to enable the new audit.
9. Execute the code under the heading for **Step 3** to add a server audit specification to the new audit.
10. Execute the code under the heading for **Step 4** to add a database audit specification to the new audit.
11. Execute the code under the heading for **Step 5** to alter the database audit specification by adding an additional action group.
12. Execute the code under the heading for **Step 6** to examine the audit metadata.
13. Execute the code under the heading for **Step 7** to examine the server audit specification metadata.
14. Execute the code under the heading for **Step 8** to examine the database audit specification metadata.
15. Execute the code under the heading for **Step 9** to remove the audit and specifications created for this demonstration.
16. Leave SQL Server Management Studio open for the next demonstration.

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Custom Audit Events



The server-level and database-level actions and action groups that SQL Server provides enable you to audit many types of events that occur in SQL Server. However, they cannot be used to audit business logic—for example, changes to an employee's salary or bonus in a human resources system.

To audit custom events such as these, you can add the USER_DEFINED_AUDIT_GROUP action group to an audit specification, and call the **sp_audit_write** stored procedure in your application code, or in a trigger, to write an event to the audit.

> **Note:** The USER_DEFINED_AUDIT_GROUP action group can be included in a server audit specification or a database audit specification.

The **sp_audit_write** stored procedure takes three parameters:

- A user defined event id (**smallint**) to identify the specific event.
- A **bit** value to track whether the event succeeded or failed.
- An information value (**nvarchar(4000)**) to describe the event.

The following code shows an example of how to call the **sp_audit_write** stored procedure from an insert trigger:

Calling sp_audit_write

```
CREATE TRIGGER HR.BonusChecker ON HR.EmployeeBonus
AFTER INSERT, UPDATE
AS

IF EXISTS (SELECT * FROM inserted WHERE bonus > 1000)
```

```
BEGIN
    EXEC sp_audit_write @user_defined_event_id = 12,
                        @succeeded = 1,
                        @user_defined_information = N'An employee bonus is over $1000';
END
```

> **Note:** You must ensure that all principals who may trigger custom audit actions have been granted EXECUTE permission on the **sys.sp_audit_write** stored procedure in the master database. The easiest way to ensure this is to grant EXECUTE permission on **sys.sp_audit_write** to **public**.

For more information on **sp_audit_write**, see *sp_audit_write (Transact-SQL)* in Microsoft Docs:

sp_audit_write (Transact-SQL)

http://aka.ms/eeq2c4

Demonstration: Using Custom Audit Events

In this demonstration, you will see how to work with custom audit events.

Demonstration Steps

1. In Solution Explorer, open the **Demo 03 - custom audit.sql** query.
2. Execute the code under the heading for **Step 1** to create a new audit.
3. Execute the code under the heading for **Step 2** to create a server audit specification including the USER_DEFINED_AUDIT_GROUP action group.
4. Execute the code under the heading for **Step 3** to call **sp_audit_write** directly.
5. Execute the code under the heading for **Step 4** to demonstrate how the custom event appears in the audit log file.
6. Execute the code under the heading for **Step 5** to create a stored procedure that uses **sp_audit_write**. The stored procedure will log a custom audit event if the discount applied to a row in the **Sales.OrderDetails** table is greater than 30 percent.
7. Execute the code under the heading for **Step 6** to call the new stored procedure twice. The second call should cause a custom audit event to be logged because the discount applied is 45 percent.
8. Execute the code under the heading for **Step 7** to examine how the custom event was recorded in the audit log file.
9. Execute the code under the heading for **Step 8** to drop the demonstration audit objects.
10. Leave SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Select the best answer

Which of the following is not a component of the SQL Server Audit architecture?

Target

Target Group

Server Audit

Database Audit Specification

Server Audit Specification

Check answer Show solution Reset

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Lesson 3: Managing SQL Server Audit

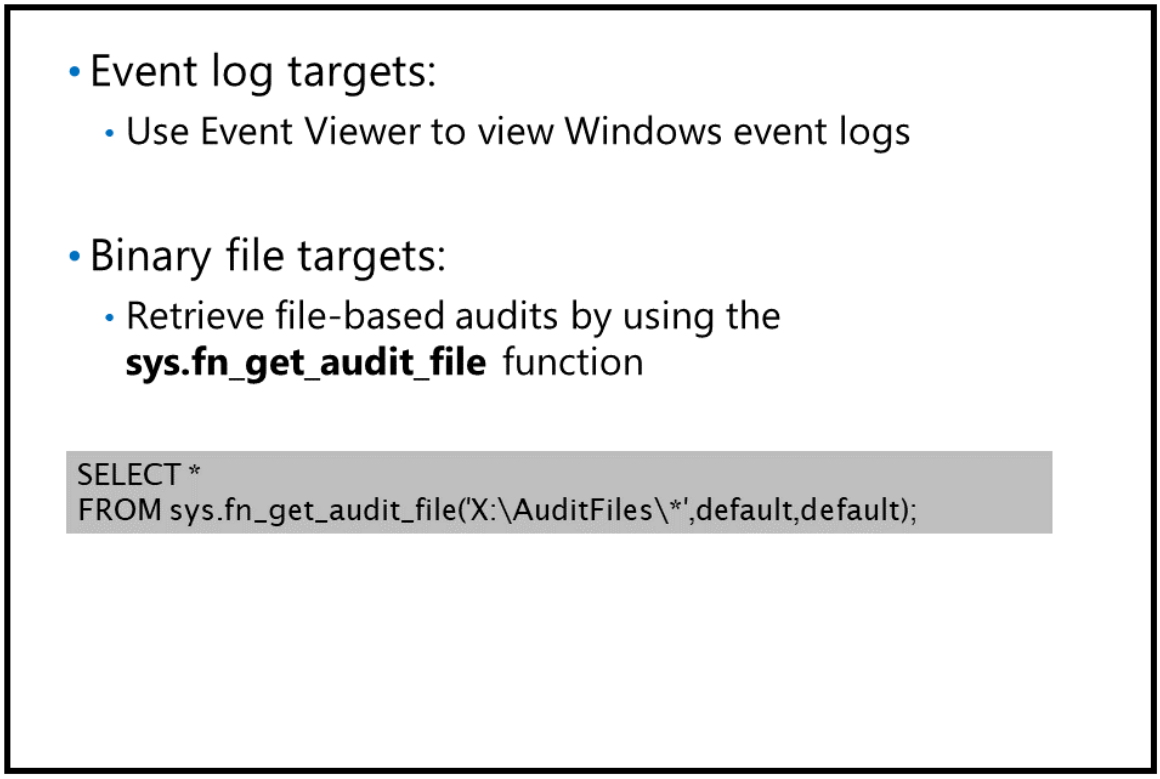
After you have configured SQL Server Audit, you need to know how to work with audit data. This lesson covers the different ways you can access audit data, and how to work with the audit record format. It also covers some potential issues you may encounter when working with servers and databases that have SQL Server Audit enabled.

Lesson Objectives

At the end of this lesson, you will be able to:

- Retrieve audit data.
- Work with the audit record structure.
- Explain potential issues you might encounter when using SQL Server Audit.

Retrieving Audit Data



The method you use to retrieve audit data will depend on the target you have specified in your audit definition.

Windows Event Log Targets

If your audit has a target of the Windows Application Log or the Windows Security Log, you can use a tool that can read Windows Event Logs, such as Event Viewer, to access audit data.

Binary File Targets

Audit files created by SQL Server Audit can be opened with the **sys.fn_get_audit_file** system table-valued function, which can read the contents of one or more audit log files and return the contents as a table that you might manipulate and filter using standard Transact-SQL statements.

The **sys.fn_get_audit_file** function takes three parameters—the file pattern, the initial file name, and the audit record offset. The file pattern can be in one of three formats:

- **<path>*** that collects audit files in the specified location. The asterisk character is a wildcard.
- **<path>\<audit name>_{GUID}** that collects all audit files that have the specified name and GUID pair.
- **<path>\<file name>** that collects a specific audit file.

This example shows how to use **sys.fn_get_audit_file** to read all the audit files in a specific directory:

sys.fn_get_audit_file—basic usage

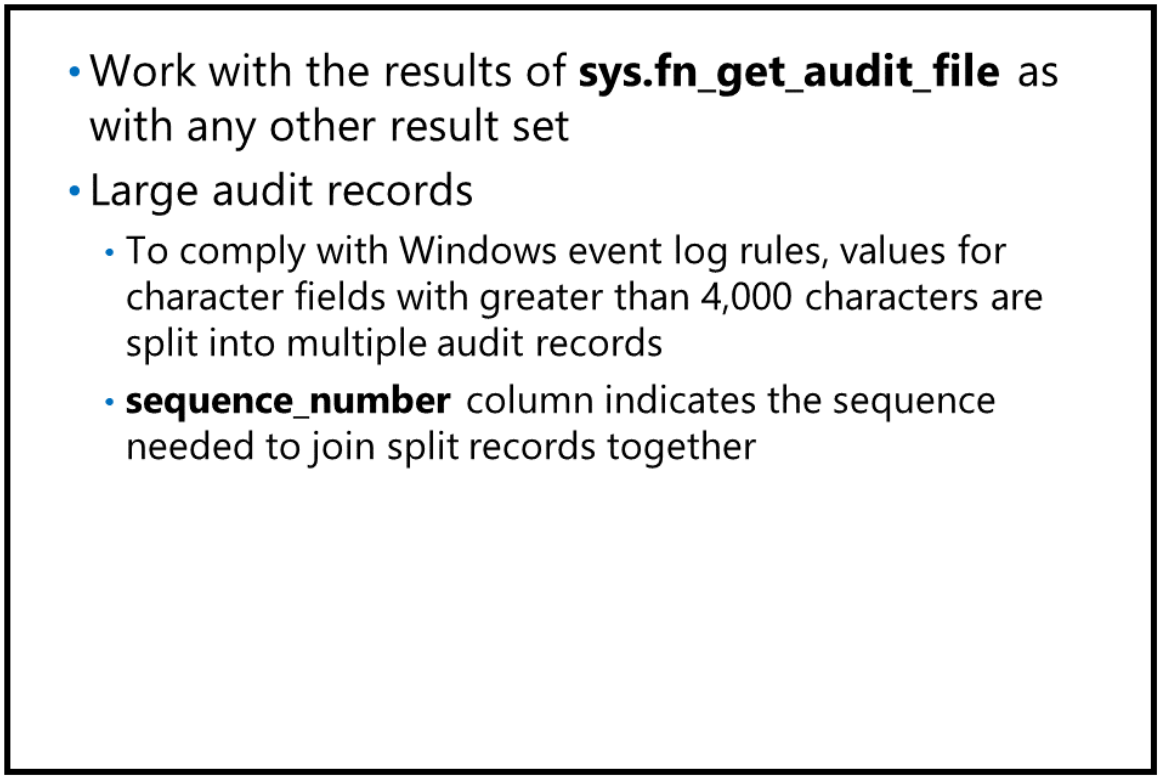
```
SELECT *
FROM sys.fn_get_audit_file('X:\AuditFiles\*',default,default);
```

For more information on **sys.fn_get_audit_file**, see *sys.fn_get_audit_file (Transact-SQL)* in Microsoft Docs:

sys.fn_get_audit_file (Transact-SQL)

http://aka.ms/p6imrw

Working with the Audit Record Structure



The table returned by the **sys.fn_get_audit_file** function returns 30 columns containing details of the circumstances in which an audit record was generated. You can work with these columns as you would with any other result set.

Full details of the columns returned by **sys.fn_get_audit_file** can be found in *sys.fn_get_audit_file (Transact-SQL)* in Microsoft Docs:

sys.fn_get_audit_file (Transact-SQL)

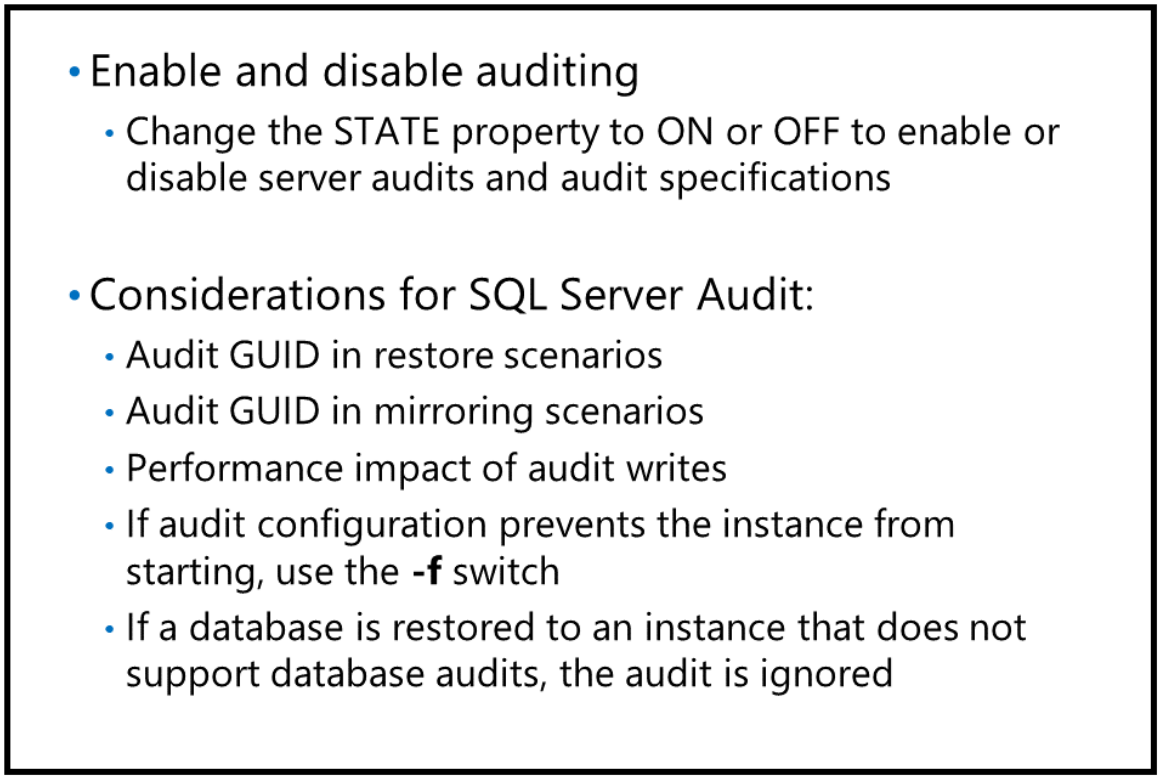
http://aka.ms/p6imrw

Large Audit Records

The audit records produced by SQL Server Audit must be formatted to fit in system event logs, and in files. Because of this requirement, the record format is limited in size by the rules related to Windows event logging systems. Character fields will be split into 4,000-character chunks that might be spread across a number of entries. This means that a single event can generate multiple audit entries and a **sequence_number** column is provided to indicate the order of multiple row entries.

Potential SQL Server Audit Issues

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



Enabling and Disabling Auditing

You can use SQL Server Management Studio to enable and disable audits or individual audit specifications. Alternatively, you can use the ALTER SERVER AUDIT, ALTER SERVER AUDIT SPECIFICATION, and ALTER DATABASE AUDIT SPECIFICATION statements to set the STATE property to ON or OFF.

You must disable audits and audit specifications before you drop them, or make any other changes to configuration.

The following code example disables the **SecurityAudit** audit:

Disabling an Audit

```
USE master;
ALTER SERVER AUDIT SecurityAudit
WITH (STATE = OFF);
```

Considerations for SQL Server Audit

There are several potential issues to consider with SQL Server Audit:

- Each audit is identified by a GUID. If you restore or attach a database on a server, SQL Server attempts to match the GUID in the database with the GUID of the audit on the server. If no match occurs, auditing will not work until you correct the issue by executing the CREATE SERVER AUDIT command to set the appropriate GUID using the AUDIT_GUID option.
- Mirrored servers introduce a similar issue of mismatched GUIDs. The mirror partner must have a server audit with the same GUID. You can create this by using the CREATE SERVER AUDIT command and supplying the GUID value to match the one on the primary server.
- If databases are attached to editions of SQL Server that do not support the same level of audit capability, the attach works but the audit is ignored.

- You should consider the performance impact of audit writes and whether you need to minimize your audit list to maximize performance.
- If insufficient disk space is available to hold audit files and you have configured an audit to shut down the server on failure, SQL Server might not start. In this situation, you may need to force entry to it by starting SQL Server in minimal configuration mode with the **-f** startup parameter.

For more information on specifying a GUID when you create a server audit, see *CREATE SERVER AUDIT (Transact-SQL)* in Microsoft Docs:

CREATE SERVER AUDIT (Transact-SQL)

http://aka.ms/mn06tw

Demonstration: Viewing the Output of SQL Server Audit

In this demonstration, you will see:

- How to view the output of an audit with a file target.
- How to view the output of an audit with a Windows event log target.

Demonstration Steps

1. In Solution Explorer, open the **Demo 04 - audit output.sql** query.
2. Execute the code under the heading for **Step 1** to create an audit with a file target.
3. Execute the code under the heading for **Step 2** to create an audit with a Windows application log target.
4. Execute the code under the heading for **Step 3** to add a specification to each audit that collects SELECT statements run against the **salesapp1.Sales** schema.
5. Execute the code under the heading for **Step 4** to execute a select statement that will be audited.
6. Execute the code under the heading for **Step 5** to examine the contents of the audit file target. Point out the most useful fields.
7. Right-click the **Start** button, and then click **Event Viewer**.
8. In Event Viewer, expand the **Windows Logs** node, then click **Application**. The audit entry will be the most recent one in the Application pane with a Source value of MSSQLSERVER. Demonstrate the entry, then close Event Viewer.
9. Execute the code under the heading for **Step 7** to remove the demonstration audit objects.
10. Leave SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Select the best answer

Which of the following is not a target for SQL Server Audit?

File

Windows Application Log

Windows Security Log

Ring Buffer

Check answer Show solution Reset

Lesson 4: Protecting Data with Encryption

Many organizations are obliged by their security compliance policies to protect data at rest by encrypting it, to mitigate the risk of the physical theft of data storage media, such as disks and backup tapes. SQL Server includes a method of encrypting data at rest—Transparent Data Encryption (TDE).

To protect the most sensitive data—for example, credit card numbers, or national identity numbers—from unauthorized access, SQL Server offers Always Encrypted, which allows data values to be encrypted by an application before they are inserted into a database. TDE can use enterprise encryption key management systems through the Extensible Key Management feature. You can also use Dynamic Data Masking to obfuscate or conceal sensitive data from users who are not authorized to see it.

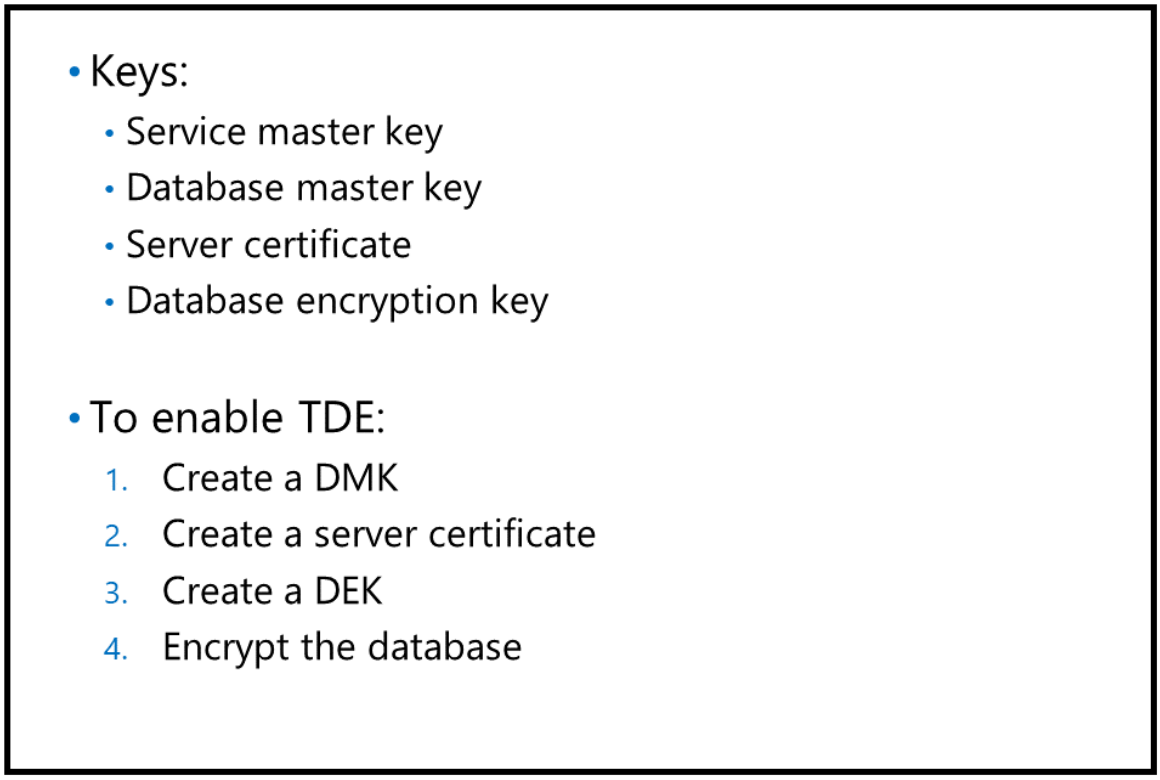
Lesson Objectives

At the end of this lesson, you will be able to:

- Explain Transparent Data Encryption.
- Move encrypted databases between SQL Server instances.
- Understand extensible key management.
- Use Always Encrypted.
- Use Dynamic Data Masking.
- Explain encryption options available in Azure SQL Database.

Transparent Data Encryption

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



TDE provides a way to secure data rendering database files unreadable without the appropriate decryption keys. It adds an extra layer of protection to your database infrastructure by encrypting data files and log files without the need to reconfigure client applications, or for additional development effort. This is because the SQL Server instance itself performs the jobs of encrypting and decrypting data, so the process is transparent to applications and users. When SQL Server writes data pages to disk, they are encrypted; when they are read into memory, they are decrypted.

> **Note:** TDE protects data at rest in database files. Data pages in the buffer pool or returned to client applications are not encrypted by TDE.

Transparent Data Encryption Keys

TDE uses the following hierarchy of keys to encrypt and decrypt data:

- **Service master key (SMK)**. The SMK is created at the time of the installation of the SQL Server instance by Setup. The SMK encrypts and protects the Database Master Key for the master database. The SMK is itself encrypted by the Windows operating system Data Protection Application Programming Interface (DPAPI).
- **Database master key (DMK)**. The DMK for the master database is used to generate a certificate in the master database. SQL Server uses the SMK and a password that you specify to generate the DMK, and stores it in the master database.

  Note: You can use a password without the SMK to generate a DMK, although this is less secure.
- **Server certificate**. A server certificate is generated in the master database, and is used to encrypt an encryption key in each TDE-enabled database.
- **Database encryption key (DEK)**. A DEK in the user database is used to encrypt the entire database.

> **Note:** When a database is configured to use TDE, CPU utilization for SQL Server may increase due to the overhead of encrypting and decrypting data pages.

Enabling TDE

TDE is only available for production use in SQL Server Enterprise edition. To enable TDE, you must perform the following steps:

1. Create a service master key in the **master** database.
2. Create a server certificate in the **master** database.
3. Create a database encryption key in the user database you want to encrypt.
4. Enable encryption for the user database.

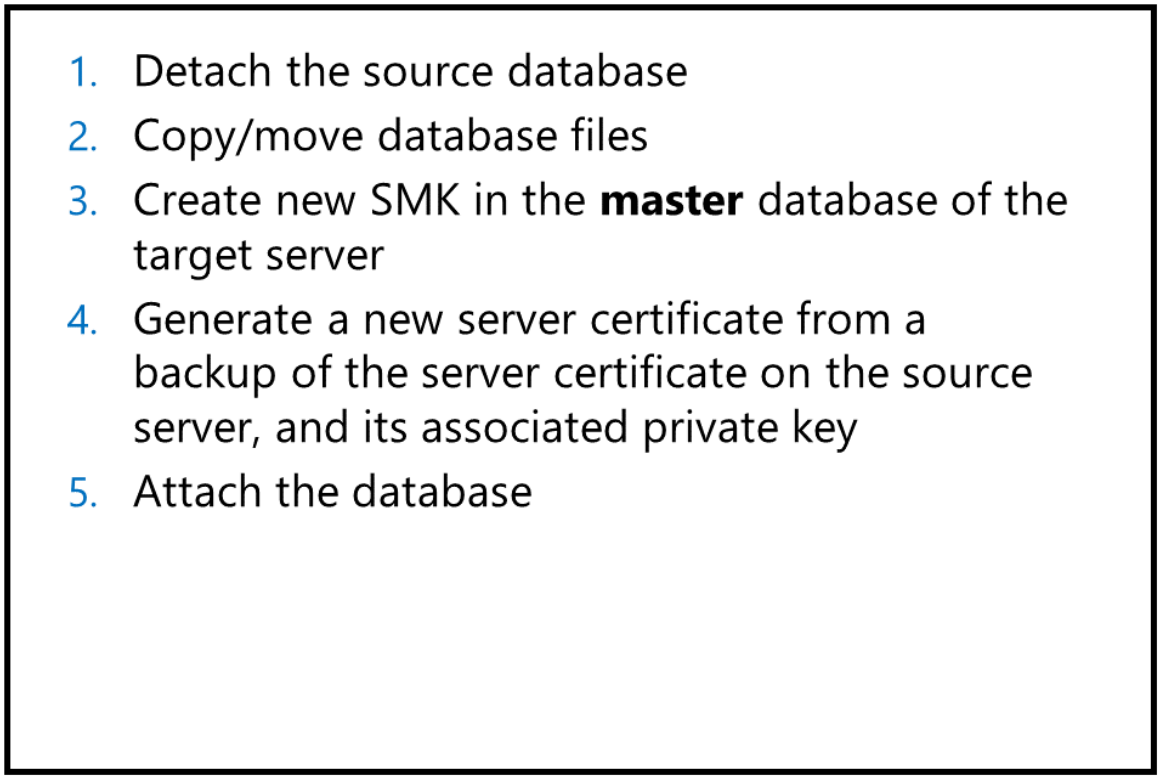
> **Additional Reading:** TDE is available in Azure SQL Database; for more information, see *Encryption with Azure SQL Database* later in this lesson.

For more information about TDE, see *Transparent Data Encryption (TDE)* in Microsoft Docs:

Transparent Data Encryption (TDE)

http://aka.ms/uy7fc2

Moving Encrypted Databases



The primary reason for encrypting a database is to prevent unauthorized access to the data it contains in the event of the data files being compromised. For this reason, you cannot attach the files from an encrypted database to another server and read its data without additional work.

If you need to move an encrypted database to another server, you must also move the associated keys and certificates. The list below describes the high level steps for moving a TDE-enabled database:

1. On the source server, detach the database that you want to move.

2. Copy or move the database files to the same location on the destination server.
3. Create a **service** master key in the **master** database on the destination server.
4. Use a CREATE CERTIFICATE Transact-SQL statement to generate a server certificate on the destination server from the backup of the original server certificate and its private key.
5. Attach the database on the destination server.

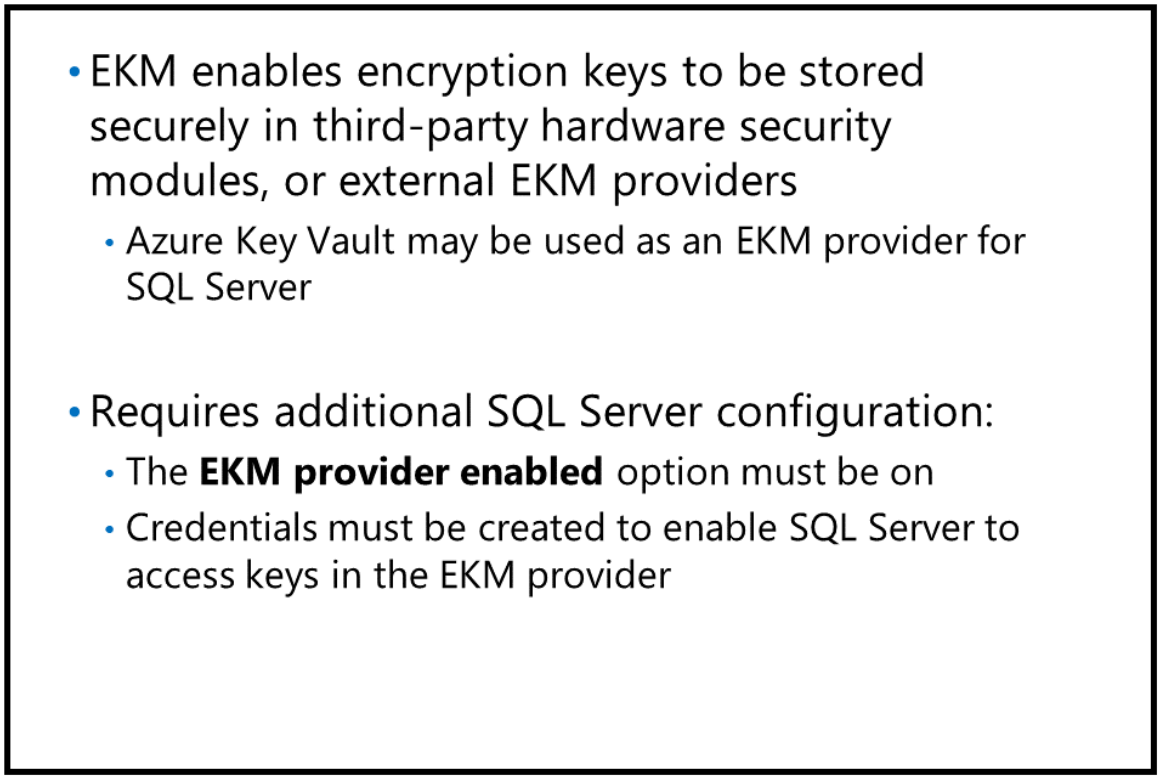
This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

For more information on moving a TDE encrypted database, see *Move a TDE Protected Database to Another SQL Server* in Microsoft Docs:

Move a TDE Protected Database to Another SQL Server

http://aka.ms/nb5dxz

Extensible Key Management



In an enterprise environment with demanding security compliance requirements, managing encryption keys at the individual database server level may not be practical. Many organizations have adopted an enterprise solution that means they can manage encryption keys and certificates using vendor-specific hardware security modules (HSM) to store keys securely. Extensible Key Management (EKM) support in SQL Server makes it possible to register modules from third-party vendors in SQL Server, enabling SQL Server to use the encryption keys stored on them. You may use Azure Key Vault as an EKM provider for SQL Server instances running on-premises or on Azure virtual machines.

EKM is only available for production use in SQL Server Enterprise edition.

For general information about managing encryption keys in SQL Server, see *SQL Server and Database Encryption Keys (Database Engine)* in Microsoft Docs:

SQL Server and Database Encryption Keys (Database Engine)

http://aka.ms/phwy4p

To enable EKM support, you must enable the server-level **EKM provider enabled** option, and then create credentials to allow SQL Server to access the EKM provider.

For information about configuring EKM support, see *Extensible Key Management (EKM)* in Microsoft Docs:

Extensible Key Management (EKM)

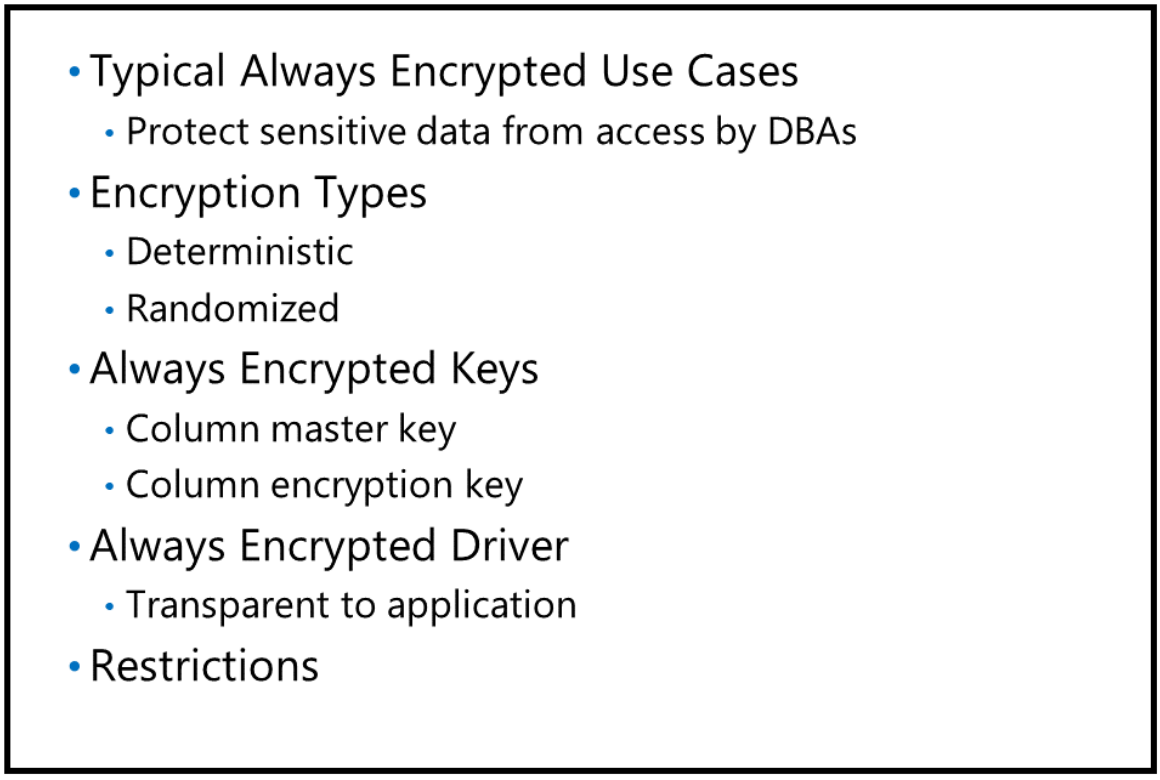
http://aka.ms/dmhvxl

For information on using Azure Key Vault as an EKM provider for SQL Server, see *Extensible Key Management Using Azure Key Vault (SQL Server)* in Microsoft Docs:

Extensible Key Management Using Azure Key Vault (SQL Server)

http://aka.ms/cbyioi

Always Encrypted



The Always Encrypted feature allows data to be transparently encrypted using a special database driver, without the encryption keys being accessible in the database. This means you can store sensitive data in databases over which you do not have complete administrative control—for example, database instances hosted by cloud services—or where data is so sensitive that it should not be accessible to SQL Server administrators. Always Encrypted is unlike TDE in the following significant ways:

- Data is encrypted both at rest and in motion. Encryption and decryption take place at the client application.
- Always Encrypted is applied at column level. TDE is applied at database level.

Typical Always Encrypted Use Cases

- Application on-premises, database on-premises. In this scenario, Always Encrypted might be used to protect sensitive data from accidental or malicious access by database administrators.
- Application on-premises, database in the cloud. In this scenario, Always Encrypted might be used to protect sensitive data from accidental or malicious access by cloud service administrators.

Encryption Types

Two types of encryption are supported by Always Encrypted:

- **Deterministic encryption.** A given plain-text value will always give the same cypher-text value. This allows filtering and grouping by ranges of encrypted values, but may allow an attacker to guess column values by analyzing patterns in the encrypted values.
- **Randomized encryption.** The cypher-text value cannot be predicted based on the plain-text value. This form of encryption is more secure, but the column value cannot be used in filters or grouping expressions.

Always Encrypted Keys

Always Encrypted relies on two types of encryption keys:

- **Column master keys**. As with master keys used in TDE, column master keys are used to create and protect column encryption keys. Column master keys must be stored in a trusted key store.
- **Column encryption keys**. Used to encrypt column data. Column encryption keys—encrypted with a column master key—are securely stored in the database.

Always Encrypted Driver

Always Encrypted encryption is carried out by an Always Encrypted-enabled driver; this makes the action of Always Encrypted transparent to client applications.

When an Always Encrypted column is referenced in a query, the driver must:

1. Retrieve the relevant column encryption key from the database.
2. Retrieve the relevant column master key from the trusted key store.
3. Use the column master key to decrypt the column encryption key.
4. Use the decrypted column encryption key to decrypt the column data.

Restrictions

There are many limitations on the use of Always Encrypted, including:

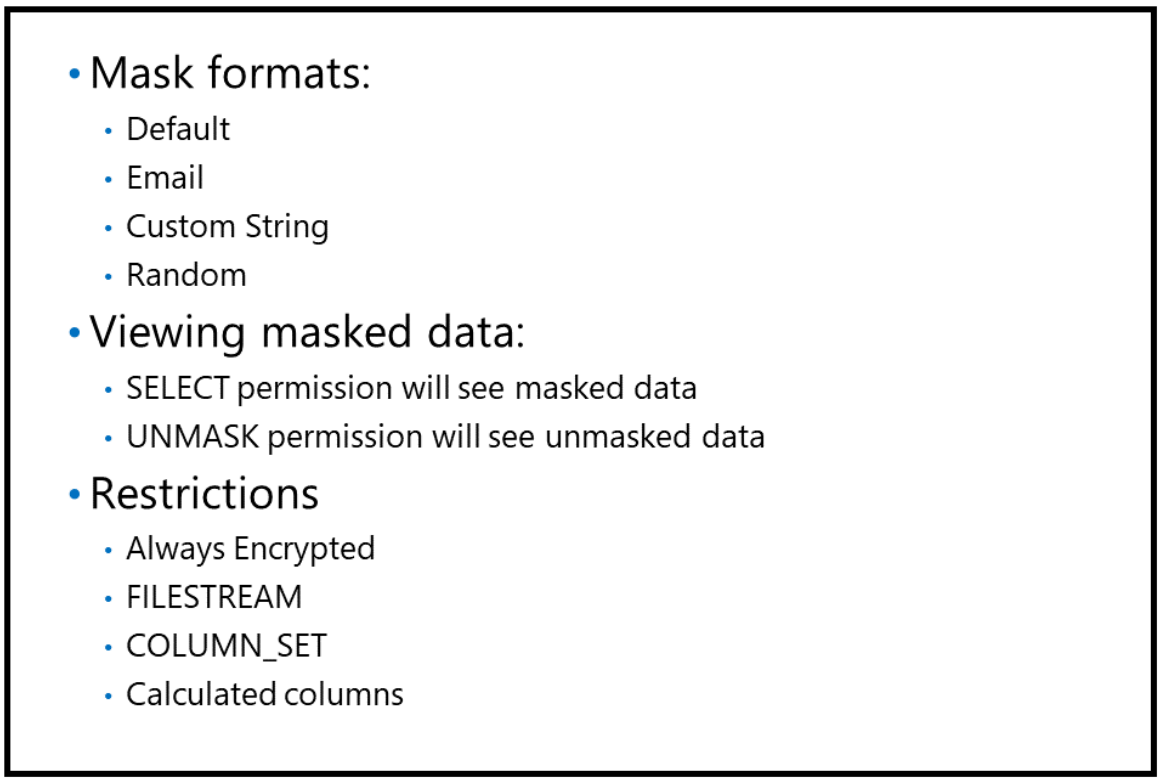
- Always Encrypted may not be used on columns of any of the following data types: **xml**, **rowversion**, **image**, **ntext**, **text**, **sql_variant**, **hierarchyid**, **geography**, **geometry**, aliased types (such as **sysname**), and user defined-types.
- Columns of any string data type (**char**, **varchar**, **nvarchar**, and so on) must use a _BIN2 collation to be eligible for Always Encrypted.

For more information on Always Encrypted, including a full list of restrictions, see *Always Encrypted (Database Engine)* in Microsoft Docs:

Always Encrypted (Database Engine)

http://aka.ms/x7koz8

Dynamic Data Masking



Dynamic data masking provides a method of concealing sensitive data from users who do not have permission to view it by masking the sensitive data values.

> **Note:** Dynamic data masking does not encrypt data at rest.

Mask Formats

Four separate data masks are available for different use cases:

- **Default**. The data is fully masked. The mask value will vary based on the data type of the masked column.
- **Email**. For string data types only. The first letter of the data is exposed; the remainder of the data is masked with “X”, and the value has the constant suffix “.com”—regardless of the actual top-level domain of the masked email addresses.
- **Custom String**. For string data types only. One or more letters at the start and end of the string are exposed. The remainder of the data is masked with a mask you can define.
- **Random**. For numeric types only. The original value is masked with a random value from within a range you specify.

Viewing Masked Data

Users with the SELECT permission on a masked column will see the masked values. Users with the additional UNMASK permission will see the unmasked data.

> **Note:** A user without the UNMASK permission, but with the UPDATE permission, will be able to update a masked column.

Restrictions

A mask cannot be specified on columns that meet the following criteria:

- Columns encrypted with Always Encrypted
- FILESTREAM columns
- COLUMN_SET columns
- Calculated columns

The following example demonstrates how to define a column masked with the default mask as part of a CREATE TABLE statement:

Dynamic Data Masking Example

```
bank_account varchar(50) MASKED WITH (FUNCTION = 'default()') NULL
```

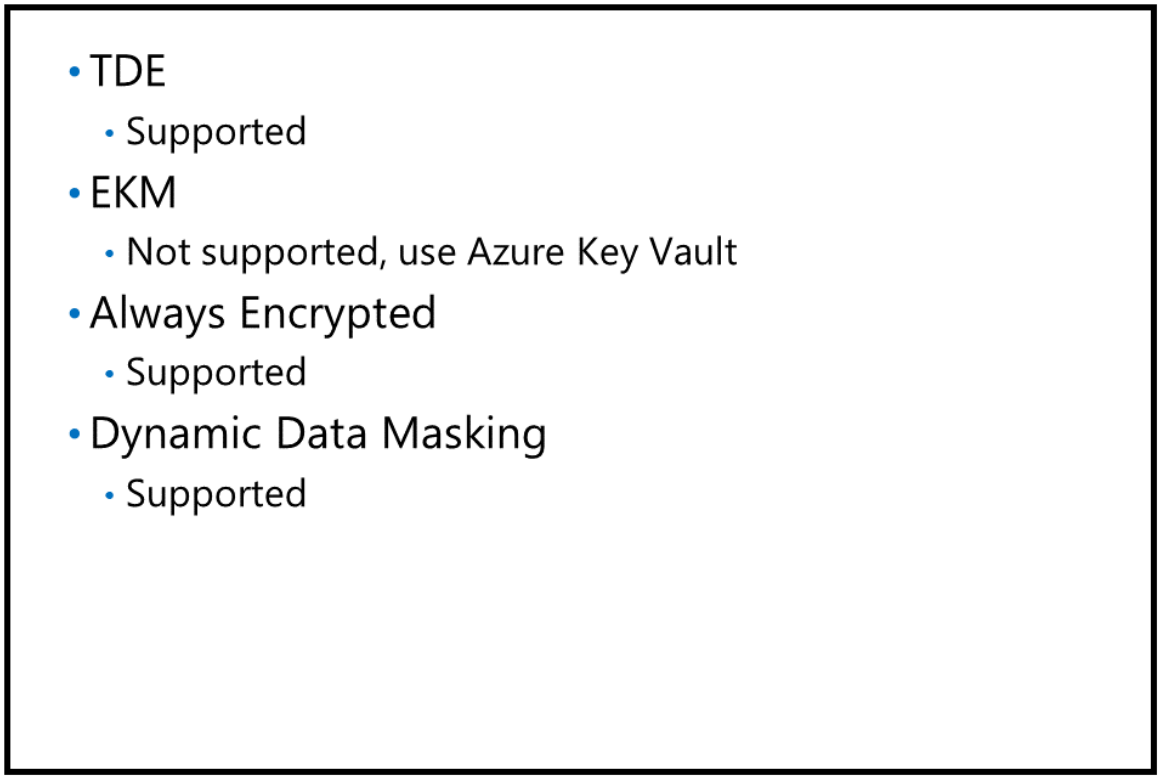
For more information on dynamic data masking, see *Dynamic Data Masking* in Microsoft Docs:

Dynamic Data Masking

http://aka.ms/s6ljbx

Encryption with Azure SQL Database

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!



TDE

TDE is supported on Azure SQL Database. It can be configured using Transact-SQL, as discussed earlier in this lesson, or it can be configured using the Azure Portal, or by using Azure PowerShell. When you use TDE on Azure SQL Database, you just mark a database as encrypted. Encryption keys and certificates are managed by Microsoft.

For more information on TDE in Azure SQL Database, see *Transparent Data Encryption with Azure SQL Database and Data Warehouse* in Microsoft Docs:

Transparent Data Encryption for Azure SQL Database and Data Warehouse

http://aka.ms/edq1g5

EKM

EKM is not supported in Azure SQL Database, but you can use the Azure Key Vault service as an EKM provider to protect your encryption keys.

Always Encrypted

Always Encrypted is supported by Azure SQL Database. It is configured in the same way as for a SQL Server instance.

Dynamic Data Masking

Dynamic data masking can be configured in Azure SQL Database using Transact-SQL statements. It can also be configured using the Azure Portal, through the database settings blade. In addition to enabling you to configure dynamic data masking, the Azure Portal will suggest columns that might be suitable candidates for a data mask.

For more information on configuring dynamic data masking through the Azure Portal, see *Get started with SQL Database Dynamic Data Masking (Azure Portal)* in the Azure documentation:

SQL Database dynamic data masking

http://aka.ms/f7c3he

Demonstration: Using Dynamic Data Masking

In this demonstration, you will see how to work with Dynamic Data Masking.

Demonstration Steps

1. In Solution Explorer, open the **Demo 05 - masking.sql** query.
2. Execute the code under the heading for **Step 1** to create a new table with data masked data, grant permission to a test user, and insert test data.
3. Execute the code under the heading for **Step 2** to demonstrate that an administrator can see unmasked data.
4. Execute the code under the heading for **Step 3** to demonstrate that a user with only SELECT permission sees the masked data. Spend some time comparing the masked output to the table definitions.
5. Execute the code under the heading for **Step 4** to add a mask to the **home_phone_number** column.
6. Execute the code under the heading for **Step 5** to demonstrate the new mask.
7. Execute the code under the heading for **Step 6** to remove the mask from the **salary** column.
8. Execute the code under the heading for **Step 7** to demonstrate that the mask on salary is no longer in place.
9. Execute the code under the heading for **Step 8** to grant the UNMASK permission to the test user. Note that it is a database-level permission.
10. Execute the code under the heading for **Step 9** to demonstrate the effect of the UNMASK permission.
11. Execute the code under the heading for **Step 10** to drop the demonstration table.
12. Close SQL Server Management Studio, without saving any changes.

Check Your Knowledge

Categorize Activity

Categorize each item by the corresponding SQL Server feature. Indicate your answer by writing the category number to the right of each item.

Encryption and decryption carried out by SQL Server
Data values obfuscated
Data encrypted at rest and in transit
Data stored in plain text
Data encrypted at rest

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Encryption and decryption take place at client application

Acts at database level

Acts at column level

TDE

Please drag items here

Always Encrypted

Please drag items here

Dynamic Data Masking

Please drag items here

Check answer Show solution Reset

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Lab: Using Auditing and Encryption

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. Following an internal security audit, the company aims to put auditing in place to track access to the database, encrypt a database at rest, and encrypt some sensitive data with Always Encrypted. You are a database administrator for Adventure Works, tasked with implementing these changes.

Objectives

After completing this lab, you will be able to:

- Work with SQL Server Audit.
- Use TDE to encrypt database files.
- Encrypt columns with Always Encrypted.

Lab Setup

Estimated Time: 90 minutes

Virtual machine: **20764C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Working with SQL Server Audit

Scenario

The **MIA-SQL** SQL Server instance will be used to host application databases, including the **salesapp1** database.

You have been asked to set up an audit that records:

- Details of a successful login attempt to the **MIA-SQL** instance.
- Details of UPDATE and INSERT statements run against the **salesapp1.HR.Employees** table.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Create a Server Audit
3. Create a Server Audit Specification
4. Create a Database Audit Specification
5. Generate Audited Activity
6. Review Audit Data
7. Disable the Audit

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After completing this exercise, you will be able to:

- Create a server audit.
- Create a server audit specification.
- Create a database audit specification.
- Retrieve audit data.

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Exercise 2: Encrypt a Column with Always Encrypted

Scenario

It has been determined that customer telephone numbers should be encrypted in the **salesapp1** database. You will implement Always Encrypted to meet this requirement.

The main tasks for this exercise are as follows:

1. Encrypt a Column
2. View Always Encrypted Data from an Application

Detailed Steps ▼

Detailed Steps ▼

Result: After completing this exercise, you will be able to implement Always Encrypted.

Exercise 3: Encrypt a Database Using TDE

Scenario

To protect all the data in the **salesapp1** database when it is at rest, you will encrypt it using TDE. You will then move the **salesapp1** database from the **MIA-SQL** instance to the **MIA-SQL\SQL2** instance.

The main tasks for this exercise are as follows:

1. Create a Service Master Key
2. Back Up the Service Master Key
3. Create and Back Up a Server Certificate
4. Create a Database Encryption Key and Encrypt the salesapp1 Database
5. Move the salesapp1 Database

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

This document belongs to DEREK PIKE. derek@systecs.co.uk No unauthorized copies allowed!

Result: After completing this exercise, you will be able to:
- Encrypt a database using TDE.
- Move an encrypted database to another SQL Server instance.

Review Question(s)

Check Your Knowledge

Select the best answer

Which type of Always Encrypted encryption will consistently encrypt the same plain text value to the same cypher text (assuming the same encryption key is used)?

Deterministic encryption

Randomized encryption

Neither of the above

Check answer Show solution Reset

Module Review and Takeaways

Best Practice: When planning to implement auditing, consider the following best practices:
- Choose the option to shut down SQL Server on audit failure. There is usually no point in setting up auditing, and then having situations where events can occur but are not audited. This is particularly important in high-security environments.
- Make sure that file audits are placed on drives with large amounts of free disk space and ensure that the available disk space is monitored on a regular basis.

Best Practice: When planning to implement database encryption, consider the following best practices:
- Use a complex password to protect the database master key for the master database.
- Ensure you back up certificates and private keys used to implement TDE, and store the backup files in a secure location.
- If you need to implement data encryption on multiple servers in a large organization, consider using an EKM solution to manage encryption keys.
- If you intend to use Always Encrypted, plan how you will store column master keys to make them accessible to applications that need to encrypt and decrypt data.

Review Question(s)

Check Your Knowledge

Discovery

You may wish to audit actions by a DBA. How would you know if the DBA stopped the audit while performing covert actions?

Show solution Reset

This document belongs to DEREK PIKE.
derek@systecs.co.uk
No unauthorized copies allowed!