Microsoft

OFFICIAL MICROSOFT LEARNING PRODUCT

20764C

Administering a SQL Database Infrastructure

Product Number: 20764C

Part Number (if applicable): X21-64456

Released: 02/2018

Welcome!

Thank you for taking our training! We've worked together with our Microsoft Certified Partners for Learning Solutions and our Microsoft IT Academies to bring you a world-class learning experience—whether you're a professional looking to advance your skills or a student preparing for a career in IT.

- **Microsoft Certified Trainers and Instructors**—Your instructor is a technical and instructional expert who meets ongoing certification requirements. And, if instructors are delivering training at one of our Certified Partners for Learning Solutions, they are also evaluated throughout the year by students and by Microsoft.

- **Certification Exam Benefits**—After training, consider taking a Microsoft Certification exam. Microsoft Certifications validate your skills on Microsoft technologies and can help differentiate you when finding a job or boosting your career. In fact, independent research by IDC concluded that 75% of managers believe certifications are important to team performance[1]. Ask your instructor about Microsoft Certification exam promotions and discounts that may be available to you.

- **Customer Satisfaction Guarantee**—Our Certified Partners for Learning Solutions offer a satisfaction guarantee and we hold them accountable for it. At the end of class, please complete an evaluation of today's experience. We value your feedback!

We wish you a great learning experience and ongoing success in your career!

Sincerely,

Microsoft Learning
www.microsoft.com/learning

Microsoft | Learning

[1] *IDC,* Value of Certification: Team Certification and Organizational Performance, *November 2006*

Acknowledgements

Microsoft Learning would like to acknowledge and thank the following for their contribution towards developing this title. Their effort at various stages in the development has ensured that you have a good classroom experience.

Aaron Johal – Content Developer

Aaron Johal is a Microsoft Certified Trainer who splits his time between training, consultancy, content development, contracting and learning. Since he moved into the non-functional side of the Information Technology business. He has presented technical sessions at SQL Pass in Denver and at sqlbits in London. He has also taught and worked in a consulting capacity throughout the UK and abroad, including Africa, Spain, Saudi Arabia, Netherlands, France, and Ireland. He enjoys interfacing functional and non-functional roles to try and close the gaps between effective use of Information Technology and the needs of the Business.

Caroline Eveleigh – Content Developer

Caroline Eveleigh is a Microsoft Certified Professional and SQL Server specialist. She has worked with SQL Server since version 6.5 and, before that, with Microsoft Access and dBase. Caroline works on database development and Microsoft Azure projects for both corporates, and small businesses. She is an experienced business analyst, helping customers to re-engineer business processes, and improve decision making using data analysis. Caroline is a trained technical author and a frequent blogger on project management, business intelligence, and business efficiency. Between development projects, Caroline is a keen SQL Server evangelist, speaking and training on SQL Server and Azure SQL Database.

David Fullstone – Content Developer

David Fullstone is a Microsoft SQL Server consultant. After pioneering mass geospatial data dissemination from SQL Server within NATO, David went on to teach big data management at the Royal School of Military Survey. After running and coordinating data management for the London Olympics, David left the military and became a full time contractor specializing in business intelligence and data management. David's database administration, development and business intelligence contracts have ranged from the military, government and banking through to rebuilding the Foreign and Commonwealth Office's global database servers.

Ed Harper – Content Developer

Ed Harper is a database developer specializing in Microsoft SQL Server. Ed has worked with SQL Server since 1999, and has developed and designed transaction-processing and reporting systems for cloud security, telecommunications, and financial services companies.

Nick Anderson – Content Developer

Nick Anderson MBCS MISTC has been a freelance Technical Writer since 1987 and Trainer since 1999. Nick has written internal and external facing content in many business and technical areas including development, infrastructure and finance projects involving SQL Server, Visual Studio and similar tools. Nick provides services for both new and existing document processes from knowledge capture to publishing.

Simon Butler – Content Developer

Simon Butler FISTC is a highly-experienced Senior Technical Writer with nearly 30 years' experience in the profession. He has written training materials and other information products for several high-profile clients. He is a Fellow of the Institute of Scientific and Technical Communicators (ISTC), the UK professional body for Technical Writers/Authors. To gain this, his skills, experience and knowledge have been judged and assessed by the Membership Panel. He is also a Past President of the Institute and has been a tutor on the ISTC Open Learning course in Technical Communication techniques. His writing skills are augmented by extensive technical skills gained within the computing and electronics fields.

Geoff Allix – Technical Reviewer

Geoff Allix is a Microsoft SQL Server subject matter expert and professional content developer at Content Master—a division of CM Group Ltd. As a Microsoft Certified Trainer, Geoff has delivered training courses on SQL Server since version 6.5. Geoff is a Microsoft Certified IT Professional for SQL Server and has extensive experience in designing and implementing database and BI solutions on SQL Server technologies, and has provided consultancy services to organizations seeking to implement and optimize database solutions.

Lin Joyner – Technical Reviewer

Lin is an experienced Microsoft SQL Server developer and administrator. She has worked with SQL Server since version 6.0 and previously as a Microsoft Certified Trainer, delivered training courses across the UK. Lin has a wide breadth of knowledge across SQL Server technologies, including BI and Reporting Services. Lin also designs and authors SQL Server and .NET development training materials. She has been writing instructional content for Microsoft for over 15 years.

Contents

Module 14: Troubleshooting SQL Server

Module 15: Importing and Exporting Data

Lab Answer Keys

About This Course

This section provides a brief description of the course, audience, suggested prerequisites, and course objectives.

Course Description

This five-day instructor-led course provides students who administer and maintain SQL Server databases with the knowledge and skills to administer a SQL server database infrastructure. Additionally, it will be of use to individuals who develop applications that deliver content from SQL Server databases.

Audience

The primary audience for this course is individuals who administer and maintain SQL Server databases. These individuals perform database administration and maintenance as their primary area of responsibility, or work in environments where databases play a key role in their primary job.

The secondary audiences for this course are individuals who develop applications that deliver content from SQL Server databases.

Student Prerequisites

In addition to their professional experience, students who attend this training should already have the following technical knowledge:

- Basic knowledge of the Microsoft Windows operating system and its core functionality.
- Working knowledge of relational databases
- Working knowledge of Transact-SQL.
- Some experience with database design

Course Objectives

After completing this course, students will be able to:

- Authenticate and authorize users
- Assign server and database roles
- Authorize users to access resources
- Protect data with encryption and auditing
- Describe recovery models and backup strategies
- Backup SQL Server databases
- Restore SQL Server databases
- Automate database management
- Configure security for the SQL Server agent
- Manage alerts and notifications
- Managing SQL Server using PowerShell
- Trace access to SQL Server
- Monitor a SQL Server infrastructure
- Troubleshoot a SQL Server infrastructure
- Import and export data

Course Outline

The course outline is as follows:

- Module 1: 'SQL Server Security' introduces SQL Server security models, logins and users.
- Module 2: 'Assigning Server and Database Roles' covers fixed server roles, user-defined server roles, fixed database roles and user-defined database roles.
- Module 3: 'Authorizing Users to Access Resources' describes permissions and the assignment of permissions,
- Module 4: 'Protecting data with Encryption and Auditing' describes SQL Server Audit.
- Module 5: 'Recovery models and Backup Strategies' describes the concept of the transaction log and SQL Server recovery models. It also introduces the different backup strategies available with SQL Server.
- Module 6: 'Backup of SQL Server databases' describes the SQL Server backup and the backup types.
- Module 7: 'Restoring SQL Server Databases' describes the restoration of databases.
- Module 8: 'Automating SQL Server management' describes how to use SQL Server agent for automation. It also explains the benefits of using master and target servers to centralize the administration of automation.
- Module 9: 'Configuring Security for SQL Server agent' describes the considerations for SQL Server agent security, including proxy accounts and credentials.
- Module 10: 'Monitoring SQL Server with Alerts and Notifications' covers the configuration of database mail, alerts and notifications.
- Module 11: 'Introduction to managing SQL Server by using PowerShell' introduces PowerShell for SQL Server.
- Module 12: 'Tracing Access to SQL Server with Extended Events' introduces how to use SQL Profiles and SQL Trace stored procedures to capture information about SQL Server. The module also describes how to use Distributed Replay to capture trace information from multiple servers and how to monitor locking.
- Module 13: 'Monitoring SQL Server' explains how to use Distributed Management Views to monitor SQL Server. It also describes configuration of data collection and the SQL Server utility.
- Module 14: 'Troubleshooting SQL server' explains the SQL Server troubleshooting methodology and discusses the most common issues that arise when working with SQL Server systems.
- Module 15: 'Importing and exporting data' covers the use of import/export wizards and explains how they relate to SSIS. The module also introduces BCP and BULK INSERT.

Course Materials

The following materials are included with your kit:

- ***Course Handbook:*** a succinct classroom learning guide that provides the critical technical information in a crisp, tightly-focused format, which is essential for an effective in-class learning experience.
    - **Lessons**: guide you through the learning objectives and provide the key points that are critical to the success of the in-class learning experience.
    - **Labs**: provide a real-world, hands-on platform for you to apply the knowledge and skills learned in the module.
    - **Module Reviews and Takeaways**: provide on-the-job reference material to boost knowledge and skills retention.
    - **Lab Answer Keys**: provide step-by-step lab solution guidance.

Additional Reading: Course Companion Content on the http://www.microsoft.com/learning/en/us/companion-moc.aspx Site*:* searchable, easy-to-browse digital content with integrated premium online resources that supplement the Course Handbook.

- **Modules**: include companion content, such as questions and answers, detailed demo steps and additional reading links, for each lesson. Additionally, they include Lab Review questions and answers and Module Reviews and Takeaways sections, which contain the review questions and answers, best practices, common issues and troubleshooting tips with answers, and real-world issues and scenarios with answers.
- **Resources**: include well-categorized additional resources that give you immediate access to the most current premium content on TechNet, MSDN®, or Microsoft® Press®.

Additional Reading: Student Course files on the http://www.microsoft.com/learning/en/us/companion-moc.aspx Site: includes the Allfiles.exe, a self-extracting executable file that contains all required files for the labs and demonstrations.

- ***Course evaluation:*** at the end of the course, you will have the opportunity to complete an online evaluation to provide feedback on the course, training facility, and instructor.
- To provide additional comments or feedback on the course, send email to mcspprt@microsoft.com. To inquire about the Microsoft Certification Program, send an email to mcphelp@microsoft.com.

Virtual Machine Environment

This section provides the information for setting up the classroom environment to support the business scenario of the course.

Virtual Machine Configuration

In this course, you will use Microsoft® Hyper-V™ to perform the labs.

 **Note:** At the end of each lab, you must revert the virtual machines to a snapshot. You can find the instructions for this procedure at the end of each lab

The following table shows the role of each virtual machine that is used in this course:

Virtual machine	Role
20764C-MIA-DC	MIA-DC1 is a domain controller.
20764C-MIA-SQL	MIA-SQL has SQL Server installed
MSL-TMG1	TMG1 is used to access the internet

Software Configuration

The following software is installed on the virtual machines:

- Windows Server 2016
- SQL2017
- Microsoft Office 2016
- SharePoint 2017

Course Files

The files associated with the labs in this course are located in the D:\Labfiles folder on the 20764C-MIA-SQL virtual machine.

Classroom Setup

Each classroom computer will have the same virtual machine configured in the same way.

Course Hardware Level

To ensure a satisfactory student experience, Microsoft Learning requires a minimum equipment configuration for trainer and student computers in all Microsoft Learning Partner classrooms in which Official Microsoft Learning Product courseware is taught.

- Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V) processor
- Dual 120-gigabyte (GB) hard disks 7200 RM Serial ATA (SATA) or better
- 16 GB of random access memory (RAM)
- DVD drive
- Network adapter
- Super VGA (SVGA) 17-inch monitor
- Microsoft mouse or compatible pointing device

- Sound card with amplified speakers

Additionally, the instructor's computer must be connected to a projection display device that supports SVGA 1024×768 pixels, 16-bit colors.

Module 1

SQL Server Security

Contents:

Module Overview

Protection of data within your Microsoft® SQL Server® databases is essential and requires a working knowledge of the issues and SQL Server security features.

This module describes SQL Server security models, logins, users, partially contained databases, and cross-server authorization.

Objectives

After completing this module, you will understand, and have a working knowledge of:

- SQL Server basic concepts.
- SQL Server connection authentication.
- User login authorization to databases.
- Partially contained databases.
- Authorization across servers.

Lesson 1
Authenticating Connections to SQL Server

Security implementation for SQL Server usually begins at the server level, where users are authenticated, based on logins, and organized into server-level roles to make it easier to manage permissions.

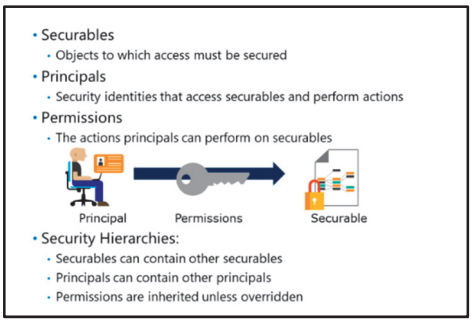
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the concepts important to understanding SQL Server security.
- Describe SQL Server authentication options and work with them.
- Understand how the Azure® SQL database firewall operates and is configured.
- Manage logins and policies in SQL Server.

Overview of SQL Server Security

Before learning how to configure security in SQL Server, it will be useful to explore some basic security concepts, and identify how they relate to SQL Server. Security is a major feature of all enterprise software systems; many of the concepts relating to security are similar across multiple systems.



Securables, Principals, and Permissions

Security is generally concerned with allowing someone or something to access a resource and to perform one or more actions on it. For example, a network administrator might need to give users access to view the contents of a folder. In more general terms, the resource on which the action is to be performed is referred to as a *securable*; the "someone or something" that needs to perform the action is referred to as a *principal*; and the configuration that allows the action to be performed is referred to as a *permission*. In the previous example, the folder is the securable, the user is the principal, and the administrator must grant the user the "read" permission on the folder.

Security Hierarchies

Security architectures are often hierarchical, primarily to simplify management of permissions. In a hierarchical security architecture, securables can contain other securables; for example, a folder can contain files, and principals can contain other principals—where users are added to a group, for example. Permissions are usually inherited, both by hierarchies of securables (for example, granting "read" permission on a folder implicitly grants "read" permission on the files it contains), and by hierarchies of principals (for example, granting "read" permission to a group implicitly grants read permission to all users who are members of that group). Generally, inherited permissions can be explicitly overridden at different hierarchy levels, to fine-tune access.

This hierarchical arrangement simplifies permission management in a number of ways:

- Fewer individual permissions need to be granted, reducing the risk of misconfiguration. You can set the general permissions that are required at the highest level in the hierarchy, and only apply explicit overriding permissions further down the hierarchy, to handle exceptional cases.

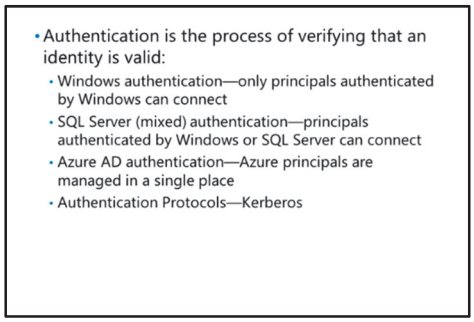
- After the permissions have been set, they can be controlled through group membership. This makes it easier to manage permissions in environments where new users arrive and existing users leave or change roles.

Best Practice: When planning a security solution, consider the following best practices:

- Provide each principal with only the permissions they actually need.
- Use securable inheritance to minimize the number of implicit permissions that must be set to enable the required level of access.
- Use principal containers, such as groups or roles, to create a layer of abstraction between principals and permissions to access securables. You can then use membership of these groups to control access to resources via the permissions you have defined. Changes in personnel should not require changes to permissions.

SQL Server Authentication

Authentication is the process of ensuring that a principal is who, or what, they state they are so they can use one or more services. To connect to the SQL Server Database Engine, the principal must supply authentication information that matches their stored credential.



Who Performs the Authentication?

When you configure SQL Server, you must choose an authentication mode for the database engine. You can choose between Windows® and mixed mode. Windows authentication is always used, but you can add further SQL Server authentication—this is then known as mixed mode.

Note: Azure Active Directory®. Azure Active Directory (Azure AD) means you can manage user authentication (and other Microsoft services) in a single, central location.

Windows Authentication

SQL Server checks the provided user name and password against the Windows user details. SQL Server does not require a password. This is the default authentication mode. Windows user security controls access to SQL Server. Access is granted based on an access token issued when the user logged in to the Windows session.

Note: Kerberos security protocol is used to provide security policies such as account locking, strong passwords, and password expiration. The Kerberos protocol is supported by SQL Server over the TCP/IP, named pipes, and shared memory communication protocols.

Because groups of users can be created in Windows, domain access administration is simplified.

Best Practice: If possible, Windows authentication should be used.

SQL Server Authentication (Mixed Mode)

Mixed mode is the addition of SQL Server authentication to Windows authentication.

SQL Server authentication requires a login when the application is started. The user name and password are stored within database tables, and so are separate from the Windows authentication. The following optional policies can be applied:

- **Change Password at Next Login**. This feature is enabled in SSQ Server Management Studio.
- **Password Expiration**. You can set a maximum age policy for passwords.
- **Windows Password Policy**. You can enforce Windows policies for passwords. For example, minimum and maximum password lengths, and other rules that define minimum standards for passwords. For example, forcing the user to include capitals, numbers, and punctuation marks in their passwords.

Note:

  - If you install SQL Server using mixed mode authentication, setup enables a SQL Server login called **sa**. It is important to create a complex password for this login because it has administrative rights at database server level.
  - If you install using Windows authentication mode, then change to mixed authentication later, this does not enable the **sa** login.

Azure Active Directory

You use Azure AD to manage user identities for connections to Azure databases in a single place. This can help you provide integrated authentication policies at both Windows and database level. For further details about using Azure Active Directory, see *Use Azure Active Directory Authentication for authentication with SQL Database or SQL Data Warehouse* in the Microsoft Azure documentation:

Use Azure Active Directory Authentication for authentication with SQL Database or SQL Data Warehouse

http://aka.ms/Wo9lg9

Protocols for Authentication

Windows authentication is typically performed by using the Kerberos protocol. The Kerberos protocol is supported with SQL Server over the TCP/IP, named pipes, and shared memory network protocols.

The SQL Server Native Access Client (SNAC) provides encrypted authentication for SQL Server logins. If SQL Server does not have a Secure Sockets Layer (SSL) certificate installed by an administrator, SQL Server generates and self-signs a certificate for encrypting authentication traffic.

Note: Encrypted authentication only applies to clients running the SQL Server 2005 version of SNAC or later. If an earlier client that does not understand encrypted authentication tries to connect, by default SQL Server will allow the connection, but the username and password will be transmitted in plain text. If this is a concern, you can use SQL Server Configuration Manager to disallow unencrypted authentication from down-level clients by setting **Force Encryption** to **Yes**.

To find information about SQL Server Authentication modes, see *Choose an Authentication Mode* in Microsoft Docs:

Choose an Authentication Mode

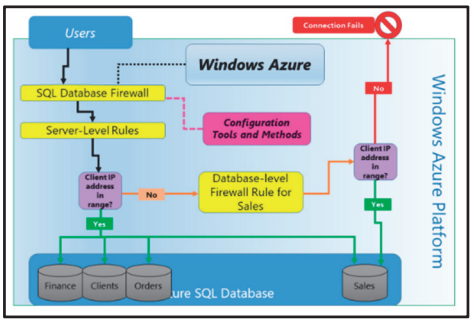
http://aka.ms/Lem9ua

Azure SQL Database Firewall

Microsoft Azure SQL Database is a cloud-based relational database service that you can use with Internet applications and SQL Server components such as SQL Server Management Studio (SSMS).

The Azure SQL Database Firewall protects your information by providing access to the IP addresses you specify.

This topic describes how you can configure the Azure SQL database firewall to allow or deny connections to your data.



Note: Azure is described in other modules of the SQL Server Database course.

For more information about Azure, see *Microsoft Azure*:

Microsoft Azure

http://aka.ms/Os6j0m

Firewall Rules

There are two levels at which you configure Azure Firewall Rules:

- **Server**. You can configure rules that are stored on the **master** database to allow principals access to your server.
- **Database**. You can configure rules to allow principals access to individual databases. These rules are stored on the specific databases. For example, you might want certain principals to be denied access to a financial database but allow access to a marketing database.

Best Practice: Microsoft recommends using database-level firewall rules. You should consider using server rules if you have many databases with the same access requirements.

Azure Database Firewall Rules

Note: Before proceeding, you need to ask your systems administrators for your outbound IP address range.

The initial state for your Azure Database Firewall is to block all access. To enable access, you should go to the Azure Portal and create one or more server-level and database-level rules containing the IP address range you are using. You should also specify whether Azure applications can connect to your Azure SQL Database server.

The following summarizes Azure Database Firewall rule configuration:

- To allow server access, you create a server-level rule containing the IP address range.
- To allow access to a specific database, you create a database-level rule configured as follows:
    - Make sure that the IP address the principal is using is outside of the one that is specified in the server-level rule.
    - Create a database-level rule using the IP address of the principal.
    - Ensure the IP address of the principal falls within the range specified by the database-level rule.

For more information on how to configure Azure firewall settings, see the *Manage firewall rules using the Azure portal* section of *Azure SQL Database server-level and database-level firewall rules* in the Microsoft Azure documentation:

 **Azure SQL Database server-level and database-level firewall rules**

http://aka.ms/iddzy1

Making a Connection Through the Azure Database Firewall

When a connection is attempted over the Internet, the following process occurs:

1. **Server Level**. If the client IP address is within a range configured in a server-level rule, the connection is allowed to your Azure SQL Database Server.
2. **Database Level**. If the client IP address is not within the server-level range, then the database-level rules are checked. If the IP address falls within an IP range linked to one of the database-level rules, then access is granted to all databases that the IP address has been assigned to.
3. **Connection Denied**. If the client IP address falls outside of both server-level and database-level rules, the connection will be denied.

 **Note: Local Device**. To access the Azure SQL Database, the firewall on your local device TCP port **1433** must allow outgoing communications.

Managing Your Azure Database Firewall

The following sections provide a summary of the methods used to manage your Azure Database Firewall.

Transact-SQL

You can execute a query from SQL Server Management Studio or the Classic Azure Portal. You need to connect to the **master** database, and then you can select, create, update or delete firewall rules.

For example, to view firewall rules using Transact-SQL:

Viewing Your Microsoft Azure Firewall Rules

```
SELECT * FROM sys.firewall_rules ORDER BY name;
```

For more information about querying Azure firewall settings using Transact-SQL, see the *Manage firewall rules using Transact-SQL* section of *Azure SQL Database server-level and database-level firewall rules* in the Microsoft Azure documentation:

Azure SQL Database server-level and database-level firewall rules

http://aka.ms/F650g1

Azure PowerShell™

To manage your firewall rules using Azure PowerShell, see the *Manage firewall rules using Azure PowerShell* section of *Azure SQL Database server-level and database-level firewall rules* in the Microsoft Azure documentation:

Azure SQL Database server-level and database-level firewall rules

http://aka.ms/Szncif

REST API

To manage your firewall rules using REST API, see the *Manage firewall rules using REST API* section of *Azure SQL Database server-level and database-level firewall rules* in the Microsoft Azure documentation:

Azure SQL Database server-level and database-level firewall rules

http://aka.ms/K1fu0d

Troubleshooting Your Azure Database Firewall

Issues that might affect your ability to connect through the Azure Database Firewall include:

- **Local Firewall Rules**. TCP Port **1433** must be allowed. If you are inside the Azure cloud boundary, you might have to create rules for additional ports.
- **Network Address Translation (NAT)**. Your local device might use a different IP address to the one you use to connect to Azure SQL Database. You can go to the portal and configure the Azure Database Firewall to use the **Current Client IP Address**.
- **Dynamic IP Addresses**. In some cases, you might have to ask your Internet Service Provider (ISP) for the range they use for your device. You can also obtain a static address.

For more detailed information about troubleshooting your Azure Database Firewall, see the *Troubleshooting the database firewall* section of *Azure SQL Database server-level and database-level firewall rules* in the Microsoft Azure documentation:

Azure SQL Database server-level and database-level firewall rules

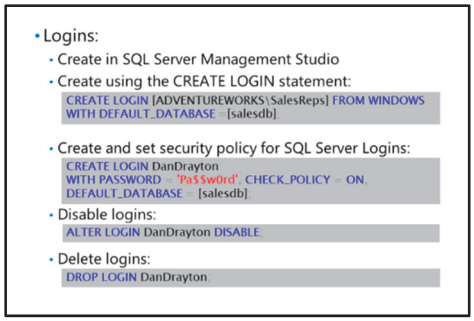
http://aka.ms/Gyqdui

For more information about ports beyond 1433 for ADO.NET 4.5, see *Ports beyond 1433 for ADO.NET 4.5* in the Microsoft Azure documentation:

Ports beyond 1433 for ADO.NET 4.5

http://aka.ms/Ppycp4

Managing Logins and Policies



You can create logins by using either Transact-SQL code or SSMS. Because this can be a very common operation, you might find it much faster, more repeatable, and accurate to use a Transact-SQL script.

Creating Logins

To create a login by using SSMS, expand the **Security** node for the relevant server instance, right-click **Logins**, and then click **New Login**. Complete the details in the **Login - New** dialog box to configure the login that you require. Alternatively, you can create logins by using the CREATE LOGIN Transact-SQL statement.

In this example, a login named ADVENTUREWORKS\SalesReps is created for the Windows group of the same name. The default database for the user will be **salesdb**. If you do not specify this option, the default database is set to master.

Creating a Windows Login

```
CREATE LOGIN [ADVENTUREWORKS\SalesReps]
FROM WINDOWS
WITH DEFAULT_DATABASE = [salesdb];
```

Note: Windows user and group names must be enclosed in square brackets because they contain a backslash character.

You create SQL Server logins in the same way. There are, however, additional arguments that are only relevant to SQL Server logins (for example, the PASSWORD argument).

The following example shows how to create a login named **DanDrayton** and assign a password of **Pa55w.rd**:

Creating a SQL Server Login

```
CREATE LOGIN DanDrayton
WITH PASSWORD = 'Pa55w.rd',
DEFAULT_DATABASE = [salesdb];
```

SQL Server Login Security Policy

In a Windows-based environment, administrators can enable policies for Windows users that enforce password complexity and expiration. SQL Server can enforce similar restrictions for SQL Server logins.

When you create a SQL Server login, you can specify the following options to control how the password policy is enforced:

- **MUST_CHANGE**: SQL Server will prompt the user to change their password the next time they log on. You must ensure that whatever client application the user will use to connect to SQL Server supports this. The default value for this setting is ON when using the user interface to create a login, but off when using a Transact-SQL CREATE LOGIN statement.
- **CHECK_POLICY = {ON | OFF}**: Setting this value to ON enforces the password complexity policy for this user. The default value for this setting is ON.

- **CHECK_EXPIRATION = {ON | OFF}**: Setting this value to ON enables password expiration, forcing the user to change their password at regular intervals. The default value for this setting is ON when using the user interface to create a login, but off when using a Transact-SQL CREATE LOGIN statement.

You can configure policy settings for a SQL Server login in SSMS, or in the CREATE LOGIN or ALTER LOGIN statement. The following code example modifies the DanDrayton login created earlier to explicitly disable policy checking:

Setting the Password Policy for a Login

```
ALTER LOGIN DanDrayton
WITH CHECK_POLICY = OFF, CHECK_EXPIRATION = OFF;
```

The full application of account policy is not always desirable. For example, some applications use fixed credentials to connect to the server. Often, these applications do not support the regular changing of login passwords. In these cases, it is common to disable password expiration for those logins.

You can reset passwords by using SSMS or the ALTER LOGIN Transact-SQL statement.

Changing a Password

```
ALTER LOGIN DanDrayton
WITH OLD_PASSWORD = 'Pa55w.rd',
PASSWORD = 'NewPa55w.rd';
```

Disabling and Deleting Logins

If logins are not being used for some time, you should disable and re-enable them later. If there is any chance that a login will be needed again in the future, it is better to disable it rather than drop it.

The following code shows how to use the ALTER LOGIN statement to disable a login:

Disabling a Login

```
ALTER LOGIN DanDrayton DISABLE;
```

You can remove logins from a server by using the DROP LOGIN statement or SSMS. If a user is currently logged in, you cannot drop their login without first ending their session.

In this example, a login is dropped from the server instance:

Dropping a Login

```
DROP LOGIN DanDrayton;
```

Demonstration: Authenticating Logins

In this demonstration, you will see how to:

- Set the authentication mode.
- Create logins.
- Manage server-level roles.
- Manage server-level permissions.

Demonstration Steps

Set the Authentication Mode

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are running, and log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod01** folder, run **Setup.cmd** as an administrator.
3. Click **Yes** when prompted to confirm that you want to run the command file, and then wait for the script to finish.
4. Start SQL Server Management Studio, and connect to the **MIA-SQL** database engine using Windows authentication.
5. In Object Explorer, right-click the **MIA-SQL** instance, and click **Properties**.
6. In the **Server Properties - MIA-SQL** dialog box, on the **Security** page, verify that **SQL Server and Windows Authentication mode** is selected, and then click **Cancel**.

Create Logins

1. In Object Explorer, expand **Security**, and expand **Logins** to view the logins that are currently defined on this server instance.
2. Right-click **Logins**, and click **New Login**.
3. In the **Login - New** dialog box, next to the **Login** name box, click **Search**.
4. In the **Select User**, **Service Account, or Group** dialog box, click **Object Types**.
5. In the **Object Types** dialog box, ensure only **Users** and **Groups** are selected, and then click **OK**.
6. In the **Select User**, **Service Account, or Group** dialog box, click **Locations**.
7. In the **Locations** dialog box, expand **Entire Directory**, click **adventureworks.msft**, and then click **OK**.
8. In the **Select User**, **Service Account, or Group** dialog box, click **Advanced**.
9. In the **Select User**, **Service Account, or Group** dialog box, click **Find Now**. This produces a list of all users and groups in the Active Directory domain.
10. In the list of domain objects, click **HumanResources_Users** (this is a domain local group that contains multiple global groups, each of which in turn contains users), and then click **OK**.
11. In the **Select User, Service Account, or Group** dialog box, ensure that **HumanResources_Users** is listed, and then click **OK**.
12. In the **Login - New** dialog box, in the **Default database** list, click **AdventureWorks**, and then click **OK**.
13. In Object Explorer, in the **Logins** folder, verify that the **ADVENTUREWORKS\HumanResources_Users** login is added.

14. Right-click **Logins**, and click **New Login**.
15. In the **Login - New** dialog box, in the **Login name** box, type **Payroll_Application**, and then click **SQL Server authentication**.
16. Enter and confirm the password **Pa55w.rd**, and then clear the **Enforce password expiration** check box (which automatically clears the **User must change password at next login** check box).
17. In the **Default database** list, click **AdventureWorks**, and then click **OK**.
18. In Object Explorer, to the **Logins** folder, verify that the **Payroll_Application** login is added.
19. Open the **CreateLogins.sql** script file in the **D:\Demofiles\Mod01 folder** and review the code it contains. This creates a Windows login for the **ADVENTUREWORKS\AnthonyFrizzell** user and the **ADVENTUREWORKS\Database_Managers** local group, and a SQL Server login named **Web_Application**.
20. Click **Execute**, and when the script has completed successfully, in Object Explorer, refresh the **Logins** folder and verify that the logins have been created.
21. On the **File** menu, click **Close**.
22. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which one of these statements is incorrect?	
Select the correct answer.	
	There are two levels at which you configure Azure Firewall Rules: Server and Database.
	You can only reset passwords using the ALTER LOGIN Transact-SQL statement.
	The trusted server application model is commonly used in large-scale enterprise applications, websites, and Internet services.
	In a Windows-based environment, administrators can enable policies for Windows users that enforce password complexity and expiration. SQL Server can enforce similar restrictions for SQL Server logins.
	To connect to SQL Server, the principal must supply information that matches the credential data held by SQL Server.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? To allow connection to the Azure Database, your local firewall rules must allow TCP Port **1344**.	

Lesson 2
Authorizing Logins to Connect to Databases

After creating a login, you should give that login access to at least one database before they can connect to the server. Generally, you only need to enable logins to access the databases that they need to work with. You can do this by creating a database user for the login in each database that it must access.

In this lesson, you will see how to create and manage database-level authorization, including database users and database roles.

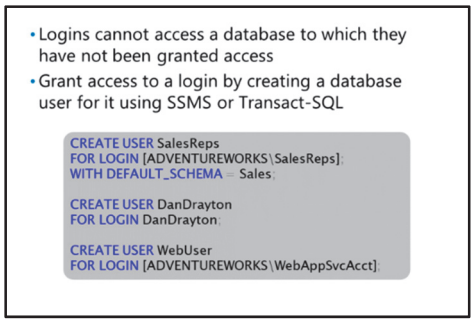
Lesson Objectives

After completing this lesson, you will be able to:

- Understand how you authorize users.
- Manage **dbo** and **guest** access rights.
- Understand how to grant access to databases.
- Authorize logins and user tokens.
- Understand and work with partially contained databases.
- Configure principals (for example, users) with passwords so that they can access the relevant databases.

Granting Access to Databases

Logins cannot connect to a database to which they have not been granted access. To grant access to a principal, you can create a database user. You can create database users by using SSMS or Transact-SQL statements.



Note: SSMS. To create a new database user in SSMS, expand the relevant database, expand the **Security** node, right-click the **Users** node, and then click **New User**. Complete the details in the **Database User - New** dialog box to configure the user you require.

You can also create database users by using the CREATE USER statement.

Creating Database Users

```
USE salesdb;

-- Example 1
CREATE USER SalesReps
FOR LOGIN [ADVENTUREWORKS\SalesReps]
WITH DEFAULT_SCHEMA = Sales;

-- Example 2
CREATE USER DanDrayton
FOR LOGIN DanDrayton;

-- Example 3
CREATE USER WebUser
FOR LOGIN [ADVENTUREWORKS\WebAppSvcAcct];
```

 **Note:** The names of Windows logins must be enclosed in square brackets "**[...]**" because they contain a backslash "****" character.

Schemas are namespaces that are used to organize objects in the database. If no default schema is specified when a user is created, then **dbo** is used.

In the above code:

- **Example 1**. The user is assigned the default schema **Sales**.
- **Example 2**. No default schema is specified, so the user will default to the **dbo** schema.
- **Example 3**. No default schema is specified, so the schema will be **dbo**. Also note that the username is different to the login with which it is associated.

You can remove users from a database by using the DROP USER statement or SSMS. However, you cannot drop a database user who owns any securable object (for example, tables or views).

Managing Mismatched Security Identifiers

When you create a SQL Server login, it is allocated both a name and a security identifier (SID). When you then create a database user for the login, details of both the name and the SID of the login are entered into the **sysusers** system table in the database. If the database is then backed up and restored onto another server, the database user is still present in the database—but there might be no login on the server that matches it. If you then create a new login with the same name and map as a user in the database, it will not work. This is because the new login has a different SID to the original login—so it cannot be added to the database.

To resolve the different SID issue, you need to update the database user to link it to the new login on the server by using the ALTER USER statement with the WITH LOGIN clause.

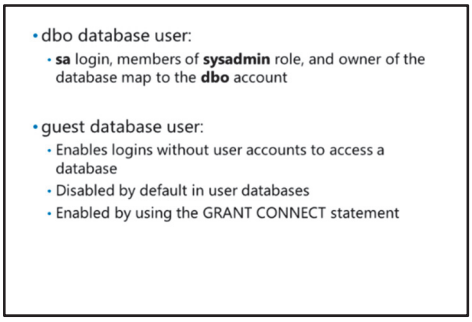
Resolving Mismatched SIDs

```
ALTER USER DanDrayton WITH LOGIN = DanDrayton;
```

This solves the issue but, if you later restore the database on the same or a different server, the problem will reoccur. A better way of avoiding the problem is by using the WITH SID clause when you create the login.

Note: Managing mismatched SIDs, orphaned users (those whose logins are disconnected when a database is moved to another SQL Server instance), impersonation and delegation are discussed in a later lesson, *Authorization Across Servers*.

Managing dbo and guest Access



While it is generally true that a login cannot access a database without having been explicitly granted access through the creation of a database user, there are two exceptions. Each SQL Server database includes two special database users—**dbo** and **guest**.

dbo User

The **dbo** user is a special user who has permissions to perform all activities in the database. Any member of the sysadmin fixed server role (including the **sa** user when using mixed mode authentication) who uses a database is mapped to the special database user called **dbo**. You cannot delete the **dbo** database user and it is always present in every database.

Database Ownership

Like other securable objects in SQL Server, databases also have owners—these are mapped to the **dbo** user.

The following example shows how you can modify the owner of a database by using the ALTER AUTHORIZATION statement:

Changing the Database Owner

```
ALTER AUTHORIZATION ON DATABASE::salesdb
TO [ADVENTUREWORKS\Database_Managers];
```

Any schema created by a login mapped to the **dbo** user will automatically have **dbo** as its owner. By default, objects within a schema have their owner set to NULL and inherit the owner of the schema in which they are defined. Owners of objects have full access to the objects and do not require explicit permissions before they can perform operations on those objects.

guest User

The **guest** user account enables logins that are not mapped to a database user in a particular database to gain access to that database. Login accounts assume the identity of the guest user when the following conditions are met:

- The login has access to SQL Server, but not the database, through its own database user mapping.
- The **guest** account has been enabled.

You can enable the **guest** account in a database to give anyone with a valid SQL Server login access to it. The **guest** username is automatically a member of the public role. (Roles will be discussed in a later module.)

A **guest** user accesses a database in the following way:

- SQL Server checks to see whether the login that is trying to access the database is mapped to a database user in that database. If it is, SQL Server grants the login access to the database as that database user.
- If the login is not mapped to a database user, SQL Server then checks to see whether the **guest** database user is enabled. If it is, the login is granted access to the database as **guest**. If the **guest** account is not enabled, SQL Server denies access to the database for that login.

You cannot drop the **guest** user from a database, but you can prevent it from accessing the database by using the REVOKE CONNECT statement. Conversely, you can enable the **guest** account by using the GRANT CONNECT statement.

Enabling and Disabling the guest Account

```
REVOKE CONNECT FROM guest;

GRANT CONNECT TO guest;
```

Note: By default, the **guest** user is enabled in the **master**, **msdb**, and **tempdb** databases. You should not try to revoke the guest access to these databases.

Authorizing Logins and User Tokens

Security tokens are used by SQL Server to work out which permissions to apply.

Security Token Service

The Security Token Service (STS) is an identity provider program that creates and issues security tokens using a variety of methods, including Kerberos. Windows Identity Foundation (WIF) allows you to build and use your own security tokens. STS allows a single sign-on for multiple services and applications—the security data is held in one place or in a chain of locations.

Briefly, instead of SQL Server directly authorizing a principal such as a client application or user, the principal is passed to an STS—such as WIF.

For example, for a Windows user attempting to connect to a SQL Server database:

1. Windows login service is redirected to WIF.
2. WIF will build a security token based on the records it holds about the user.
3. The principal (or login service) then presents the token to SQL Server to verify that the principal is authorized for access and, if correct, will create a session for the principal.
4. Once the connection is made, then SQL Server will create a list of all security tokens for each Windows group the user belongs to. This is a recursive procedure so that nested Windows groups are identified.

5. If the user context changes to another database, a further list of security tokens is made. In other words, SQL Server and WIF will make sure the user has access to the new database. If the user has permission to access the new database, then the security token is added to the token list. As in the previous step, the collection and building of the token list is repeated recursively.
6. The two lists contain security tokens representing all of the principals for the user, and each contains a record of deny or grant permissions. This is the *Active Permission Set*.

Strong encryption is used throughout this process to avoid compromising security.

For more information about WIF, see *Windows Identity Foundation* in the Microsoft .NET documentation:

Windows Identity Foundation

https://aka.ms/U42695

Viewing Security Tokens

You can view security tokens using **sys.login_token** and **sys.user_token** system views. The following code examples describe and show how each of these views is used:

Login tokens are created to authorize login credentials. The following code returns a row for each server principal that is part of the login token:

Viewing Login Tokens

```
SELECT * FROM sys.login_token;
```

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

sys.login_token (Transact-SQL)

http://aka.ms/Qmwvd2

User tokens are created to authorize user credentials. The following code returns a row for each server principal that is part of the user token:

Viewing User Tokens

```
SELECT * FROM sys.user_token;
```

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

sys.user_token (Transact-SQL)

http://aka.ms/Lx2e6w

Demonstration: Authorizing Logins and User Tokens

In this demonstration, you will see how to:

- Create a server role
- Create a login
- Alter server roles
- Create a user

View the results

Demonstration Steps

1. In SQL Server Management Studio, open the **Security Tokens Demo.sql** script file in the **D:\Demofiles\Mod01** folder. Note that in some cases, to view the results in SQL Server Management Studio, right-click the containing node in Object Explorer, and then click **Refresh** to update the objects.
2. Select the code under **Step A**, and then click **Execute**.
3. Select the code under **Step B**, and then click **Execute**.
4. In Object Explorer, under **MIA-SQL** under **Security**, expand **Server Roles** to view the new server roles.
5. Select the code under **Step C**, and then click **Execute**.
6. In Object Explorer, under **MIA-SQL** under **Security**, under **Logins**, view the new login.
7. Select the code under **Step D**, and then click **Execute**.
8. Select the code under **Step E**, and then click **Execute**.
9. In Object Explorer, under **MIA-SQL** expand **Databases**, expand **AdventureWorks**, expand **Security**, expand **Roles**, and then expand **Database Roles** to view the new database roles.
10. Select the code under **Step F**, and then click **Execute**.
11. In Object Explorer, under **MIA-SQL** under **Databases**, under **AdventureWorks**, under **Security**, expand **Users** to view the new user.
12. Select the code under **Step G**, and then click **Execute**.
13. Select the code under **Step H**, and then click **Execute** to view the current user and login tokens. The code is executed in the security context of the login created for this demonstration. Notice that tokens relating to the database user, both the **MyExtDatabaseRole** and **MyDatabaseRole** database roles, and the **public** database role, are linked to the user.
14. Select the code under **Step I**, and then click **Execute** to remove all changes.
15. On the **File** menu, click **Close**.
16. Keep SQL Server Management Studio open for the next demonstration.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? You can view security tokens using **sys.login_token** and **sys.user_token** system views.	

Lesson 3
Authorization Across Servers

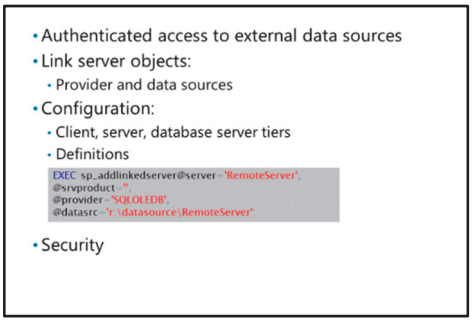
This lesson will describe multiserver authorization.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand linked servers and their security.
- Describe and understand the "double-hop" problem.
- Understand the differences between impersonation and delegation.
- Work with mismatched Security IDs.

Linked Servers Security



You can execute distributed queries on OLE DB data sources that are external to the current SQL Server instance. For example, you might wish to execute Transact-SQL against tables in another SQL Server instance, Microsoft Excel®, Access® or other database products, such as MySQL or Oracle. You use the OLE technology to manage connections between diverse data sources.

As the data sources might be out of your security domain, you need to ensure that you understand how linked servers operate and how to secure any connections between them.

The benefits of linked servers are as follows:

- **External Data Access**. Enables access to data that is external to the SQL Server instance.
- **Diverse External Data Sources**. You can work with many types of data sources within your organization.
- **Standardized Method**. The way you interact with other data sources is standardized.

Linked Server Objects

The following objects are required to use linked servers with OLE DB:

- **Provider**. This is a dynamic-link library (DLL) file that manages connections to a specific data source.
- **Data Source**. The object that contains the data you wish to work with. They are usually databases but they can also be other data sources, such as spreadsheets.

Note:

- The Native Client OLE DB provider is the SQL Server provider.
- SQL Server has been tested against the Native Client OLE DB provider and other providers.

The Open Database Connectivity (ODBC) provider can enable secure connections to many other data sources.

Linked Server Configuration

So that distributed queries reach the correct data source and return the data, the deployment of linked servers requires the building of three tiers:

1. **Client Tier**. The application that requires the data. This is where you will create the distributed queries and present them to your SQL Server instance.
2. **Server Tier**. Your SQL Server instance directs the query to the relevant database via the OLE DB provider DLLs—for example, a SQL Server database, Oracle and Microsoft Access. ODBC enables connections to many other data sources, including Microsoft Excel.
3. **Database Server Tier**. These are the actual data sources. Note that you can use multiple SQL Server databases for your distributed query.

 **Note:** Third-party OLE DB providers must have a read and execute SQL Server service account for all folders and subfolders in which the **.dll** is located.

Linked Server Definitions

During the configuration of a linked server, you need to register connection and source data information with the SQL Server instance. Once set up, data sources can be referenced in distributed queries by a single name.

You can use SQL Server Management Studio to add, change and remove linked server connection details:

- **Create**. To set up a linked server, you use Object Explorer. Open the SQL Server instance, right-click **Server Objects**, select **New**, then select **Linked Server**. You can then enter details in the **New Linked Server** dialog.
- **View or Amend**. Select the linked server, right-click and select **Properties**. The **Linked Server Properties** dialog is displayed for you to change or view details.
- **Delete**. To delete a linked server, you right-click the server and click **Delete**.

Stored procedures and catalog views can also be used to manage linked server connections:

- **sp_addlinkedserver**. Creates a linked server definition.
- **sys.servers**. You can query the system catalog views to return linked server details.
- **sp_dropserver**. Deletes a linked server definition and can be used to remove references to remote servers. Note that you can also drop remote logins using this stored procedure, as shown in the following example.
- **sp_addlinkedsrvlogin** and **sp_droplinkedsrvlogin**. These stored procedures allow you to map and remove logins between your SQL Server instance and other server security accounts. See below for further details.

The following code example shows how the **sp_addlinkedserver** and **sp_dropserver** can be used to create and drop a linked server for **RemoteServer**:

Create and Drop a Linked Server Connection

```
-- Create linked server
USE master
GO
EXEC sp_addlinkedserver
@server='RemoteServer',
@srvproduct='',
@provider='SQLOLEDB',
@datasrc='r:\datasource\RemoteServer';
GO

--Drop linked server and remove the default login mapping
USE master
GO
EXEC sp_dropserver 'RemoteServer', ‘droplogins’;
GO
```

 **Note: Fully Qualified Naming**. The following four-part name format is required in a distributed query when referring to a linked server: linked_server_name.catalog.schema.object_name.

For more information about linked servers, see *Linked Servers (Database Engine)* in Microsoft Docs:

 **Linked Servers (Database Engine)**

http://aka.ms/jmfec7

Linked Server Security

For linked server connections to operate, a login mapping must be created. You can use the system stored procedure **sp_addlinkedsrvlogin** to map logins. You can drop a connection using **sp_droplinkedsrvlogin**. Note that you can also use **sp_dropserver** to remove mapped logins as described above.

When you run a stored procedure that queries a remote server or a distributed query linked server, there must be a mapping between the requesting instance and the data sources.

 **Note:** Windows Authentication Mode should be used if possible. However, some data sources might not allow Windows Authentication Mode—for these, you can set up a Windows authenticated login locally and map this to a specific login on the data source.

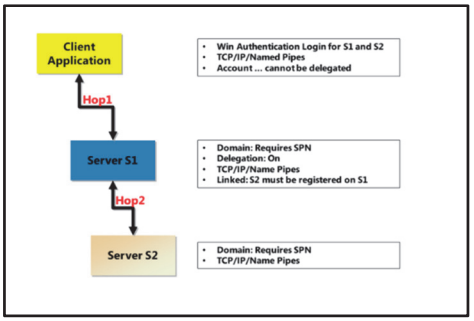
The following code example shows how you can map and drop login mappings to **RemoteServer** for the **LocalUserName** and **RemoteUserName**:

Create and Drop a Mapped Login to a Remote Server

```
-- Add a remote login mapping
USE master
GO
EXEC sp_addlinkedsrvlogin 'RemoteServer', 'false', 'LocalDomain\LocalUserName',
'RemoteUserName', 'Pa55w.rd';
GO

-- Connect all local logins to a linked server
USE master
GO
EXEC sp_addlinkedsrvlogin 'RemoteServer';
GO

-- Drop login mapping
USE master
GO
EXEC sp_droplinkedsrvlogin 'RemoteServer', NULL;
GO
```

You can test linked server connections in SQL Server Management Studio by right-clicking on the linked server and clicking **Test Connection** in the **Properties** dialog box.

Typical "Double-Hop" Problem



A common feature in applications is that they need to access data from back-end data sources. Authentication can fail because the second and subsequent connections do not have the correct authentication details. Typically, this might occur when a distributed query is executed from a linked database that accesses other servers using mapped logins.

For example, consider a Windows user, **USER1**, on an application that connects to a SQL Server instance, **SQL1**. **SQL1** has a linked server **SQL2** complete with mapped logins. **USER1** wants to execute a distributed query against **SQL2** but the query fails because **USER1** Windows Authentication fails on **SQL2**.

This is the "double-hop" problem. To solve this, we need to consider how to make sure that **SQL2** is aware that **USER1** has Windows Authentication that allows the query to execute. Delegation is required so that **SQL1** forwards the credentials of **USER1** to **SQL2**. Delegation allows **SQL1** to impersonate the user so that **SQL2** will run the distributed query.

Delegation Requirements

Client. The following list summarizes the client (**USER1**) requirements:

- **Access Permissions**. **USER1** must have Windows Authenticated logins for **SQL1** and **SQL2**.
- **Client Device**. Must use TCP/IP or named pipes.
- **Active Directory**. The Active Directory **Account is sensitive and cannot be delegated** property must not be checked for **USER1**.

Middle Tier. The following list summarizes the **SQL1** requirements:

- **Domain Admin**. **SQL1** requires a registered Service Principal Number (SPN).
- **Delegation**. The active SQL Server account must be trusted for delegation. See below for more details.
- **Server Device**. Must use TCP/IP or named pipes.
- **Linked Server**. **SQL2** must be configured as a linked server on **SQL1**.

Database Tier. The following list summarizes the **SQL2** requirements:

- **Server Device**. Must use TCP/IP or named pipes.
- **Domain Admin**. **SQL2** requires a registered SPN.

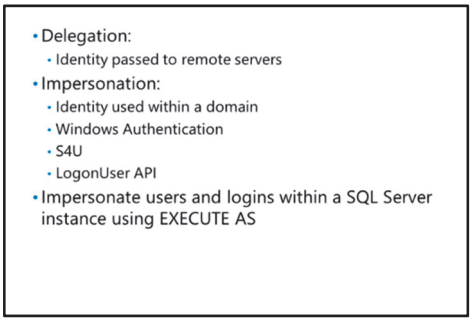
For more information about double-hop, see *Configuring Linked Servers for Delegation* in Microsoft TechNet:

Configuring Linked Servers for Delegation

https://aka.ms/Vfwuxk

Impersonation vs. Delegation

Impersonation is a method used by services to control the client access to resources within a domain—for example, local files. Delegation is a method that allows a client to execute and access resources on other devices—for example, shared files.



Delegation

You have already seen how delegation works in the previous topic *Typical "Double-Hop" Problem*. In this case, a SQL Server request used a forward identity to execute on a remote server.

Impersonation

Impersonation is carried out on the remote server in a similar way to if it were executed on the local server.

There are three methods used for impersonation:

- **Windows Authentication**. This uses a Windows identity token using Kerberos or the Security Support Provider Interface. The supplied identity token can then be cached on the service for use in impersonation.
- **Service-For-User (S4U)**. Impersonation is carried out using a Windows identity token obtained from Kerberos extensions. S4U is used when clients use non-Windows authentication.

Note: The process account must have the **Act as part of the operating system** access right.

- **LogonUser API**. An identity token can be requested and obtained from the **LogonUser** API. You can use this Windows API to access remote services but delegation trust is not used.

Impersonation in a Database Engine Instance

To test security configurations, you could impersonate a server login or a database user with the EXECUTE AS Transact-SQL command. Using EXECUTE AS switches the security context of the connection to that of the login or user specified in the statement. To return to the connection's original security context, you use the REVERT Transact-SQL command.

The following examples demonstrate how to use EXECUTE AS to impersonate a login, and then a user:

EXECUTE AS Example

```
SELECT SUSER_SNAME(), USER_NAME();

EXECUTE AS LOGIN = 'MyLogin'
      SELECT SUSER_SNAME(), USER_NAME();
REVERT

EXECUTE AS USER = 'DemoUser'
      SELECT SUSER_SNAME(), USER_NAME();
REVERT

SELECT SUSER_SNAME(), USER_NAME();
```

For more information about using EXECUTE AS, see *EXECUTE AS (Transact-SQL)* in Microsoft Docs:

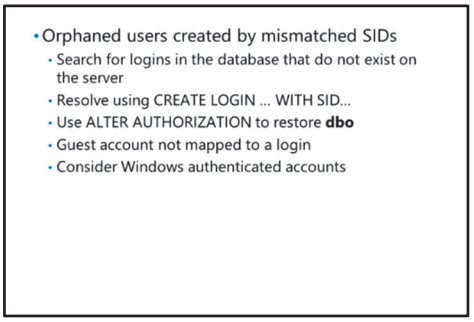
EXECUTE AS (Transact-SQL)

http://aka.ms/Kho6sq

Working with Mismatched Security Identifiers

Mismatched security identifiers (SIDs) can occur when you move a database from one SQL Server instance to another. SIDs on the source SQL Server instances will not match the SIDs on the destination database; the users are orphaned if their logins are affected. Users with mismatched SIDs might not have usable logins.

This is because logins are associated with SIDs and the SIDs on the destination SQL Server instance might not be the same as those recognized by the moved database. If the destination is in a different domain than the source database, then mismatched SIDs and orphaned users might occur.



Note: It is recommended to use Windows authentication whenever possible.

Resolving Orphaned Users

The following steps describe how you can resolve mismatched IDs and orphaned users:

1. **Search for Orphaned Users**. On the destination server, run the following code to return a list of orphaned users:

This Transact-SQL returns any orphaned users for **MyMovedDatabase**:

Orphaned Users

```
USE <MyMovedDatabase>;
GO;
SELECT dp.type_desc, dp.SID, dp.name AS user_name
FROM sys.database_principals AS dp
LEFT JOIN sys.server_principals AS sp
    ON dp.SID = sp.SID
WHERE sp.SID IS NULL
    AND authentication_type_desc = 'INSTANCE';
```

2. **Resolve Orphaned Users**. If any orphaned users are returned, you need to create logins for them with Transact-SQL as in the following code example:

To resolve orphaned users, you use the CREATE LOGIN statement with the SID for the orphaned user:

Resolve Orphaned Users

```
-- This code will relink the orphaned user
USE master
GO
CREATE LOGIN [<domainName\loginName>]
WITH SID = <SID>;
GO
```

3. **Database Owner (dbo)**. You can use Transact-SQL to relink an orphaned **dbo**:

To change **dbo** to **MyUser** in the destination database, you use the ALTER AUTHORIZATION statement.

Change dbo

```
ALTER AUTHORIZATION ON DATABASES::MyDatabase TO MyUser;
GO
```

Guests and Windows Group Members

The guest account is not mapped to a SQL Server login.

A SQL Server login that was sourced from a Windows account can access a database if the account is part of a Windows group and is also a database user.

Demonstration: Working with Mismatched Security IDs

In this demonstration, you will see how to:

- Test for orphaned users.
- Fix broken logins.
- Show that the logins have been corrected.

Demonstration Steps

1. In SQL Server Management Studio, open the **MismatchedIDs.sql** script file in the **D:\Demofiles\Mod01** folder.
2. Select the code under **Step A**, and then click **Execute** to run the orphaned users report in the **TSQL** database. Two users should be returned. Note the SID for the **appuser1** user.
3. Select the code under **Step B**, and then click **Execute** to demonstrate that an **appuser** login exists, but has a different SID to that referenced by the **appuser1** user.

4. Select the code under **Step C**, and then click **Execute** to repair the **appuser1** user by linking it to the **appuser** login.
5. Select the code under **Step D**, and then click **Execute** to demonstrate that **appuser1** is no longer an orphaned user. Note the SID for the **reportuser1** user.
6. Select the code under **Step E**, and then click **Execute** to create the **reportuser** login with a defined SID value. The SID matches the SID returned in the orphaned users report for **reportuser1**.
7. Select the code under **Step F**, and then click **Execute** to demonstrate that no orphaned users remain.
8. On the **File** menu, click **Close**.
9. Keep SQL Server Management Studio open.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? You can query the sys.missing_sids table to identify orphaned users in a database.	

Lesson 4
Partially Contained Databases

The containment feature in SQL Server reduces the reliance a database has on the SQL Server instance that hosts it. This is very useful for optimizing the moving of a database to another SQL Server instance. This lesson discusses the containment feature, how it is deployed, and the considerations for using partially contained databases.

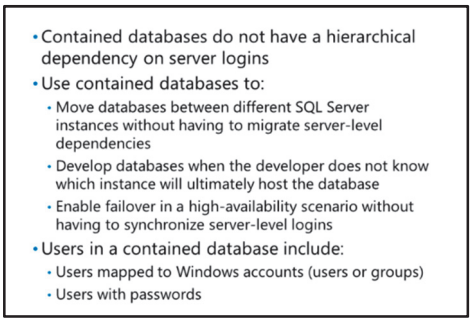
Lesson Objectives

After completing this lesson, you will be able to:

- Understand partially contained databases.
- Understand the considerations for using partially contained databases.
- Create a partially contained database.

Introduction to Partially Contained Databases

Typically, databases in a SQL Server instance have dependencies on server-level resources—most typically logins, but often on other resources too. This relationship between databases and the server instance on which they are hosted enables a hierarchical approach to security and management. However, there are some scenarios where it would be useful to completely isolate a database and its management from the server on which it resides. For example:



- When it is necessary to move databases between different SQL Server instances; entities that are external to the database, such as logins, are not moved along with the database.
- When a database is in development and the developer does not know which instance will ultimately host the database.
- When a database that participates in an Always On availability group is mirrored on multiple server instances, and it is useful to be able to failover to a secondary instance without having to synchronize the server-level logins required to access the database.

A contained database is one that is hosted on an instance of SQL Server, but which has no dependencies on the server instance. Because there are no dependencies, you can move the database between servers, or use it in availability group scenarios, without having to consider external factors, such as logins.

Containment Levels

Noncontained databases rely on their SQL Server instance for many features such as metadata, authentication, collations, and resources. Most SQL Server databases are noncontained. Partially contained databases have fewer dependencies on the hosting SQL Server instance. Partially contained databases are mostly managed separately from the database engine instance. Partially contained databases allow the use of some uncontained resources that are managed by the SQL Server instance and are outside of the particular database boundary.

Note: There is one further level of database containment: full containment. Fully contained databases hold all the metadata, data objects, data, database object logins, user accounts, and other resources they need to operate—this is usually held in the SQL Server Instance. These databases are fully stand-alone. Full containment is not supported in SQL Server 2017.

The following table provides a summary of terms and their definitions:

Term	Description
Database Boundary	The boundary between a partially contained database and other databases, and the SQL Server instance.
Contained Attribute	Refers to an element that is held within the database boundary.
Uncontained Attribute	An element that is not held within the database boundary.
Contained User	This type of user is authenticated by the partially contained database.
Windows Principals	The partially contained database will use Windows to authenticate these users—they are trusted and do not require logins to be set up in the **master** database.

Characteristics of Contained Databases

Contained databases include the following characteristics that isolate them from a server instance:

- Contained databases store the metadata that defines the database. This information is usually stored only in the master database but, in a contained database, it is also stored in the contained database itself.
- All metadata uses the same collation settings.
- The database contains users and can authenticate those users without reference to SQL Server logins. Authentication can be performed by the database itself, or by trusting users that have been authenticated by Windows.

Enabling Database Containment

To use contained databases, you must enable contained database authentication at the level of the server instance. You can do this either by using the **sp_configure** stored procedure to enable the contained databases authentication option, or by setting the **Enable Contained Databases** option to **True** on the **Advanced** page of the **Server Properties** window in SSMS.

Creating and Converting Partially Contained Databases

After the SQL Server instance has been enabled for containment, you can use either Transact-SQL or SSMS to enable or disable containment on new or existing databases.

In SSMS, you toggle containment using the **Containment Type** drop-down in the **Options** page of the **Database Properties** dialog box.

The following code shows how you can create a new partially contained database using CREATE Transact-SQL:

Creating Partially Contained Database

```
CREATE DATABASE MyDatabase CONTAINMENT = PARTIAL;
```

You can also change the containment level of an existing database; you can use the CONTAINMENT parameter of the ALTER DATABASE Transact-SQL statement to enable or disable containment.

The following code shows how to enable or disable containment in an existing table:

Alter Containment

```
-- Convert a database to partial containment
ALTER DATABASE MyDatabase SET CONTAINMENT = PARTIAL;

-- Disable off containment
ALTER DATABASE MyDatabase SET CONTAINMENT = NONE;
```

Contained Users

After creating a contained database, you can create contained users for that database. These users can be one of two types:

- Users with associated password. These users are authenticated by the database; the user's password is stored in the database metadata.
- Users that are mapped to Windows user accounts. These users exist only in the database, with no associated server-level login, and do not require the user to maintain a separate password. Instead of performing its own authentication, the database trusts Windows authentication.

You create contained users by using the CREATE USER statement in the context of a contained database.

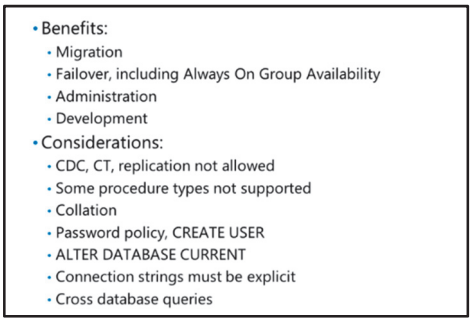
For more information about contained databases, see *Contained Databases* in Microsoft Docs:

Contained Databases

http://aka.ms/F30zfd

Considerations for Using Partially Contained Databases

This topic explores considerations for using partially contained databases.



Benefits of Using Partially Contained Databases

Moving a Database to Another Host

The primary use of partially contained databases is to simplify moving the database to another SQL Server instance—for example, quick and simple failover for business critical databases. During the movement of a noncontained database to a new host, it is possible to break the mapping between users and their logins, creating orphaned users. As logins and users are held on the contained database, this is less likely to happen.

Note: To facilitate moving contained databases, you need to document the external elements that are required for the database to function outside of the current SQL Server instance. For example, a related database might need to be noted, in addition to the relationship details. Similarly, external settings need to be identified and documented.

Administration

To manage database settings for a noncontained database, an administrator will need **sysadmin** privilege to change instance level settings. In a partially contained database, the administrator will have more control over their own database. Administrators might still require access to noncontained elements.

Partially Contained Databases and Always On

You can use Always On Availability Groups to optimize failover. Partially contained databases are more independent than uncontained databases on the SQL Server instance. As contained users can connect directly to the contained database, and then to the failover database, availability is improved. The uncontained elements can be documented and scripted, making recovery faster and more efficient. Contained elements are moved with the database. You can find out more about Always On Availability Groups in *Overview of Always On Availability Groups (SQL Server)* and *Prerequisites, Restrictions, and Recommendations for Always On Availability Groups* in Microsoft Docs:

Overview of Always On Availability Groups (SQL Server)

http://aka.ms/Hgybtz

Prerequisites, Restrictions, and Recommendations for Always On Availability Groups

http://aka.ms/O18k7a

Database Development

Development of new databases will take place away from the live environment. By using partially contained databases, the developer can evaluate the effect that instance level elements might have on the database.

Considerations When Using Partially Contained Databases

Some of the features and issues mentioned in the following list might not be drawbacks in partially contained databases and could be deployed or fixed by release. However, when designing or using partially contained databases, they should be considered and possibly documented for migration, failover or other database movements.

The following considerations need to be made when designing, developing, or using partially contained databases:

1. **Cross Database Queries**. Users with the same name and password on different partially contained databases are not the same user.
2. **ALTER DATABASE**. Use the current option, ALTER DATABASE CURRENT.
3. **Change Data Capture (CDC)**. CDC is not supported in partially contained databases.
4. **Change Tracking (CT)**. CT is not supported in partially contained databases.
5. **Password Policy**. CREATE USER does not support bypassing the password policy. If the partially contained database does not have a password policy, this can cause problems after migration.
6. **Synonyms**. Some external dependencies might be omitted and cause issues when a partially contained database is moved.
7. **Connection Strings**. You must specify the database name in application connection strings.
8. **Numbered Procedures**. Not supported in partially contained databases.
9. **Temporary Procedures**. Not supported in partially contained databases.
10. **Replication**. Not operational in partially contained databases.

11. **Collation**. Partially contained databases do not rely on **tempdb** to hold collation details in the SQL Server instance. Therefore, code that uses explicit collation might need to be altered to use CATALOG_DEFAULT and not DATABASE_DEFAULT. Schema-bound objects are dependent on collation changed built-in functions.
12. **IntelliSense**. An SSMS connection to a partially contained database with a contained user login does not wholly support this feature.
13. **SQL Server Data Tools (SSDT)**. SSDT is not aware of containment and will not inform you if containment is not adhered to.

For further information about contained databases, see *Contained Database Users – Making Your Database Portable*, *Contained Database Collations*, and *Migrate to a Partially Contained Database* in Microsoft Docs:

Contained Database Users - Making Your Database Portable

http://aka.ms/aaew1r

Contained Database Collations

http://aka.ms/Cw48th

Migrate to a Partially Contained Database

http://aka.ms/Udr653

Demonstration: Creating a Partially Contained Database

In this demonstration, you will see how to:

- Check the server instance for containment.
- Inspect code to enable or disable containment.
- Create a partially contained database.

Demonstration Steps

View the Containment Value

1. In SQL Server Management Studio, open **ContainedDatabase.sql** in the **D:\Demofiles\Mod01** folder.
2. Select the code under **Step A**, and then click **Execute**. Note that the value returned is '**1**' as containment should be enabled.
3. Select the code under **Step B**, and then click **Execute**. Note that the **value_in_use** is '**0**' (containment is disabled). To confirm this:
    a. In Object Explorer, right-click **MIA-SQL**, and then click **Properties**.
    b. In the **Server Properties - MIA-SQL** dialog box, on the **Advanced** page, note the **Enable Contained Databases** attribute is **False**, and then click **Cancel**.

4. Select the code under **Step C**, and then click **Execute**. Note that the **value_in_use** is set back to '**1**'. To confirm this:
   a. In Object Explorer, right-click **MIA-SQL**, and then click **Properties**.
   b. In the **Server Properties - MIA-SQL** dialog box, on the **Advanced** page, note the **Enable Contained Databases** attribute is **True**, and then click **Cancel**.
5. Select the code under **Step D**, and then click **Execute**.
6. In Object Explorer, under **MIA-SQL**, right-click **Databases**, and then click **Refresh** to view a list of databases, including the new partially contained database.
7. Select the code under **Step E**, and then click **Execute**. The new users will be returned from the SELECT statement.
8. In Object Explorer, under **MIA-SQL**, under **Databases**, expand **PClientData**, expand **Views**, and expand **System Views**, right-click **sys.database_principals**, and then click **Select Top 1000 Rows**. Confirm the new users have been created in the list of contained users.
9. In the **ContainedDatabase.sql** tab, select the code under **Step F**, and then click **Execute**.
10. Close SQL Server Management Studio, without saving any changes.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or False? The following code can be used to add **SalesMan1** to the **master** database: USE master GO CREATE USER SalesMan1 WITH PASSWORD = 'Pa55w.rd' GO	

Check Your Knowledge

Question	
Which option does not apply to partially contained databases?	
Select the correct answer.	
	You can use the ALTER statement to convert a noncontained database to a partially contained database.
	CDC and CT are not supported in partially contained databases.
	Numbered and temporary procedures are supported in partially contained databases.
	Replication is not operational in partially contained databases.

Lab: Authenticating Users

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler and retailer of cycle products. You are a database administrator for Adventure Works, tasked with managing access to the **MIA-SQL** SQL Server instance by creating and configuring logins and database users.

Objectives

After completing this lab, you will be able to:

- Create logins authenticated by SQL Server authentication and Windows authentication.
- Create database users.
- Resolve login issues.
- Configure security for restored databases.

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Create Logins

Scenario

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

You have been asked to create the following logins:

- A login for the ADVENTUREWORKS\WebApplicationSvc Active Directory user account.
- A login for the ADVENTUREWORKS\IT_Support Active Directory group account.
- A SQL Server login for the Sales Support application. The login should have the following properties:
  - User name: **SalesSupport**
  - Password: **Pa55w.rd**

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Verify the Authentication Mode
3. Create Logins Based on Windows Authentication
4. Create Logins Based on SQL Server Authentication

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab01\Starter** folder as Administrator.

▶ Task 2: Verify the Authentication Mode

1. Review the exercise scenario, and determine the appropriate SQL Server authentication mode for the requirements.
2. Ensure that the authentication mode for the **MIA-SQL** SQL Server instance is set appropriately to support the requirements.

▶ Task 3: Create Logins Based on Windows Authentication

1. Review the exercise scenario, and determine the required logins that can use Windows authentication.
2. Create the required logins, using the following guidelines:
   - Create the minimum number of logins required to meet the requirements.
   - All logins should use **AdventureWorks** as the default database.

▶ Task 4: Create Logins Based on SQL Server Authentication

1. Review the exercise scenario, and determine the required logins that can use SQL Server authentication.
2. Create the required login, using the following guidelines:
   - SQL Server logins should use the password **Pa55w.rd** and be subject to password policy restrictions. However, their passwords should not expire, and they should not be required to change the password when they next log in.
   - Use **AdventureWorks** as the default database.
3. Leave SSMS open for the next exercise.

Results: After this exercise, you should have verified the authentication modes supported by the **MIA-SQL** instance, and created three logins.

Exercise 2: Create Database Users

Scenario

To enable the logins you have created to access the **AdventureWorks** database, you must create users in the database and map them to the logins.

The main tasks for this exercise are as follows:

1. Use Automatic User Creation and Mapping
2. Create a User and Map It to a Login
3. Create a User Using Transact-SQL

▶ Task 1: Use Automatic User Creation and Mapping

1. In SSMS, use the **User mapping** page of the properties for the **ADVENTUREWORKS\WebApplicationSvc** to grant the login access to the **AdventureWorks** database.
2. Use SSMS Object Explorer to verify that a user called **ADVENTUREWORKS\WebApplicationSvc** has been created in the **AdventureWorks** database.

▶ Task 2: Create a User and Map It to a Login

1. Use SSMS to create a user called **ServiceUser** in the **AdventureWorks** database. Map the user to the **SalesSupport** login.
2. Using SSMS, check the properties of the **SalesSupport** login to verify that it is mapped to the **ServiceUser** user.

▶ Task 3: Create a User Using Transact-SQL

- In SSMS, start a new query and connect it to the **AdventureWorks** database. Write a Transact-SQL statement to create a database user called **ITSupport** that is mapped to the **ADVENTUREWORKS\IT_Support** login and defaults to the **dbo** schema.

Results: At the end of this exercise, you will have created three database users and mapped them to the logins you created in the previous exercise.

Exercise 3: Correct Application Login Issues

Scenario

Users of the LegacySales application have reported that the application is showing an error message stating that it is unable to connect to the database. The administrator of the LegacySales application tells you that the application configuration file shows the login **LegacySalesLogin**, with the password **t0ps3cr3t**, is used by the application to connect to the **MIA-SQL** server. No database name is specified in the configuration file; the application should be connecting to the **AdventureWorks** database.

Your task is to investigate and correct the issues with the **LegacySalesLogin**.

For more information about troubleshooting this issue, see *MSSQLSERVER_18456* in Microsoft Docs:

MSSQLSERVER_18456
http://aka.ms/N2n2j6

The main tasks for this exercise are as follows:

1. Carry Out an Initial Test
2. Enable the Login
3. Change the Login Password
4. Change the Default Database

▶ Task 1: Carry Out an Initial Test

1. Use the **sqlcmd** command-line application to test a connection to the MIA-SQL instance using the credentials supplied by the application administrator.
2. Check the SQL Server log to determine the cause of the failure.
3. Why does the login fail?

▶ Task 2: Enable the Login

1. Use SSMS to enable the **LegacySalesLogin** login, then retest using **sqlcmd**. What error message is displayed?
2. Check the SQL Server error log for more details of the failure. What does the log message indicate is the cause of the login failure?

▶ Task 3: Change the Login Password

- Use SSMS to change the password of the **LegacySalesLogin** login to **t0ps3cr3t**, then retest using **sqlcmd**. What error message is displayed?

▶ Task 4: Change the Default Database

- Use SSMS to change the default database of the **LegacySalesLogin** login to **AdventureWorks**, and add a user mapping from the login to the **AdventureWorks** database. Retest using **sqlcmd** to confirm that you have resolved the issue. Close any command prompt windows that are open from your testing. Leave SSMS open for the next exercise.

Results: After completing this lab, you will be able to:

Correct application login issues.

Exercise 4: Configure Security for Restored Databases

Scenario

The **InternetSales** database was recently restored to the **MIA-SQL** instance from a backup taken on another database server. You need to verify that security is correctly configured for the restored database. Two logins should have access to the **InternetSales** database:

- **ADVENTUREWORKS\WebApplicationSvc**
- **InternetSalesApplication**

The main tasks for this exercise are as follows:

1. Verify Database Users
2. Run the Orphaned Users Report
3. Repair the Orphaned User

▶ Task 1: Verify Database Users

- Using SSMS, verify that users exist in the **InternetSales** database with the following names:
  - **ADVENTUREWORKS\WebApplicationSvc**
  - **InternetSalesApplication**

▶ Task 2: Run the Orphaned Users Report

1. Write and execute a Transact-SQL query to run the orphaned users report in the **InternetSales** database.
2. Which user(s) are orphaned?

▶ Task 3: Repair the Orphaned User

1. Write and execute a query to use sp_change_users_login to repair the orphaned **InternetSalesApplication** user. Because a login of the same name as the user exists, the simplest method to achieve this is to use **Autofix** as the value for the **@Action** parameter.
2. Rerun the orphaned user report to verify that no orphaned users remain.
3. Close SSMS without saving any changes.

Results: At the end of this exercise, the **ADVENTUREWORKS\WebApplicationSvc** and **InternetSalesApplication** logins will have access to the **InternetSales** database.

Question: In the last exercise of the lab, why was the **ADVENTUREWORKS\WebApplicationSvc** user not reported as an orphaned user?

Module Review and Takeaways

In this module, you have learned about the core concepts of SQL Server security, including logins and database users. You have learned how to work with cross-database security, and how to work with databases that have different levels of containment.

Review Question(s)

Check Your Knowledge

Question	
What happens when a login does not have access to its default database and is used to open a connection to a SQL Server instance?	
Select the correct answer.	
	The login can connect to SQL Server, but an error message is reported.
	The login cannot connect to SQL Server.
	The login is automatically disabled.
	The login is automatically granted access to its default database.

Module 2

Assigning Server and Database Roles

Contents:

Module Overview

Using roles simplifies the management of user permissions. With roles, you can control authenticated users' access to system resources based on each user's job function—rather than assigning permissions user-by-user, you can grant permissions to a role, then make users members of roles.

Microsoft® SQL Server® includes support for security roles defined at server level and at database level.

Note: The types of roles available to you will vary, depending on whether you are using a full installation of SQL Server (running on-premises or in the cloud) or Azure® SQL Database. Azure SQL Database does not offer server-level roles.

For a comparison of the security features offered by SQL Server and Azure SQL Database, see *Security Center for SQL Server Database Engine and Azure SQL Database* in Microsoft Docs:

Security Center for SQL Server Database Engine and Azure SQL Database

http://aka.ms/rxh35d

Note: Server and database roles should be assigned permissions on the principle of least privilege; each role should hold the least permissions possible for it to function normally.

Objectives

After completing this module, you will be able to:

- Describe and use server roles to manage server-level security.
- Describe and use fixed database roles.
- Use custom database roles and application roles to manage database-level security.

Lesson 1
Working with Server Roles

Roles defined at server level are used to control access to server-level permissions. This lesson covers server-level permissions, the purposes of the built-in server roles, and how to create server-level roles that are customized to your requirements.

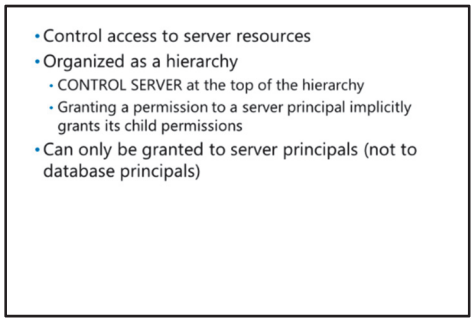
Lesson Objectives

After completing this lesson, you will be able to:

- Describe server-scoped permissions.
- Explain typical server-scoped permissions.
- Give an overview of fixed server roles.
- Explain the **public** server role.
- Create user-defined server roles.

Server-Scoped Permissions

Permissions at the server level generally relate to administrative actions, such as creating databases, altering logins, or shutting down the server. Some fundamental permissions, such as CONNECT SQL (which permits a login to connect to the SQL Server Database Engine), are also managed at server level.



You can view a complete list of server-scoped permissions using the system function **sys.fn_builtin_permissions**:

sys.fn_builtin_permissions Server

```
SELECT *
FROM sys.fn_builtin_permissions('SERVER')
ORDER BY permission_name;
```

Server permissions are organized as a hierarchy; when you grant a server-level permission to a server principal (either a login or a role), this implicitly also grants the child permissions of the granted permission to the principal.

The topmost node in the hierarchy of server-level permissions (and therefore the server permission that encompasses all other server permissions) is CONTROL SERVER.

You can visualize the hierarchy of server permissions with the following query:

Server Permissions Hierarchy

```
WITH recCTE
AS
(
      SELECT  permission_name, covering_permission_name AS parent_permission,
                  1 AS hierarchy_level
      FROM sys.fn_builtin_permissions('SERVER')
      WHERE permission_name = 'CONTROL SERVER'

      UNION ALL

      SELECT  bp.permission_name, bp.covering_permission_name,
                  hierarchy_level + 1 AS hierarchy_level
      FROM recCTE AS r
      CROSS APPLY sys.fn_builtin_permissions('SERVER') AS bp
      WHERE bp.covering_permission_name = r.permission_name
)
SELECT * FROM recCTE
ORDER BY hierarchy_level, permission_name;
```

Server permissions can only be granted to server principals (logins and server roles).

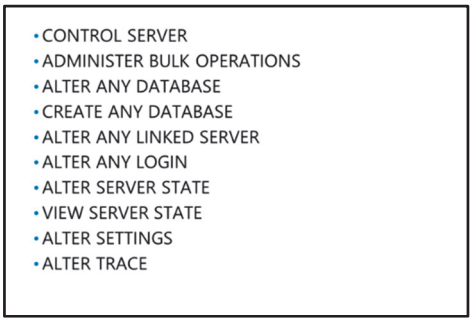
For more information on **sys.fn_builtin_permissions**, see *sys.fn_builtin_permissions (Transact-SQL)* in Microsoft Docs:

sys.fn_builtin_permissions (Transact-SQL)

http://aka.ms/uk6ymp

Typical Server-Scoped Permissions

Some examples of server-scoped permissions that you might typically want to grant include:



- CONTROL SERVER

  This permission implicitly grants all other server-level permissions.
  CONTROL SERVER is not exactly equivalent to membership of the **sysadmin** fixed server role; some system stored procedures and functions require membership of the **sysadmin** role.

- ADMINISTER BULK OPERATIONS

  This permission grants access to bulk insert operations.

- ALTER ANY DATABASE

  This permission grants administrative control over all databases on a SQL Server instance. It implicitly grants the CREATE ANY DATABASE permission.

- CREATE ANY DATABASE

  This permission grants the right to create new databases.

- ALTER ANY LINKED SERVER

  This permission grants administrative control over all linked servers.

- ALTER ANY LOGIN

  This permission grants administrative control over any login, including logins for linked servers.

- ALTER SERVER STATE

  This permission grants permission to execute DBCC commands that affect the contents of the buffer pool (DBCC FREE*CACHE and DBCC SQLPERF). It implicitly grants the VIEW SERVER STATE permission.

- VIEW SERVER STATE

  This permission grants SELECT permissions on server-level dynamic management objects.

- ALTER SETTINGS

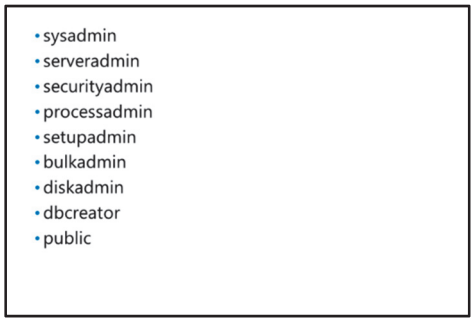
  This permission grants EXECUTE permissions on **sp_configure** and the RECONFIGURE command.

- ALTER TRACE

  This permission grants administrative control over SQL Trace.

Overview of Fixed Server Roles

Fixed server roles are preconfigured roles that correspond to common administrative areas of responsibility for a SQL Server installation. They are provided for convenience of use, and for backward compatibility with previous versions of SQL Server.



Each fixed server role is granted a group of permissions relevant to the function of the role; the permissions granted to a fixed server role cannot be changed. With the exception of the **public** role, granting a fixed server role to a server principal also grants that server principal with the permission to add other principals to the role.

The following table shows a list of the fixed server-level roles with a description of the permissions granted to role members:

Fixed Server-Level Role	Description
sysadmin	This role grants permissions to perform any action on the server. You should limit membership of this role as far as possible.
serveradmin	This role grants permissions to configure server-wide settings and to shut down the server.
securityadmin	This role grants permissions to manage logins. This includes the ability to create and drop logins and the ability to assign permissions to logins. Members of this role can grant and deny server-level permissions to other users and grant and deny database-level permissions to other users on any database to which they have access. Because of the ability to assign permissions to other users, membership of this role should be limited as much as possible. This role should be treated as equivalent to **sysadmin**.
processadmin	This role grants permissions to terminate sessions running on the SQL Server instance.

Fixed Server-Level Role	Description
setupadmin	This role grants permissions to manage linked servers.
bulkadmin	This role grants permissions to execute the BULK INSERT statement.
diskadmin	This role grants permissions to manage disk files.
dbcreator	This role grants permissions to manage databases.
public	Every user is a member of **public** and this cannot be changed. This role does not initially grant any administrative permissions. Though you can add permissions to this role, this is not advisable because the permissions would be granted to every user.

To keep to the principle of least privilege, you should avoid using fixed server roles as far as possible. Unless a fixed server role holds exactly the permissions required for a server principal, you should create a user-defined server role with only the permissions that the principal requires.

The exception to this is the **sysadmin** role. The CONTROL SERVER permission is not directly equivalent to membership of the **sysadmin** role, and there are more than 150 system stored procedures and functions that explicitly check for membership of **sysadmin** before executing.

 **Note:** Unlike in some earlier versions of SQL Server, in SQL Server 2017 the **BUILTIN\administrators** and Local System (**NT AUTHORITY\SYSTEM**) accounts are not automatically added as members of the **sysadmin** role, although you can add them manually if required. Note that this does not affect the ability of local administrators to access the database engine when it is in single-user mode.

To view membership of server-level roles, you can query the **sys.server_role_members** system view:

Role Membership

```
SELECT spr.name AS role_name, spm.name AS member_name
FROM sys.server_role_members AS rm
JOIN sys.server_principals AS spr
ON spr.principal_id = rm.role_principal_id
JOIN sys.server_principals AS spm
ON spm.principal_id = rm.member_principal_id
ORDER BY role_name, member_name;
```

For more information on fixed server roles and the server-level permissions assigned to them, see *Server-Level Roles* in Microsoft Docs:

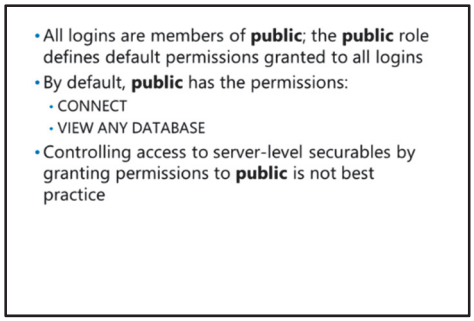
 **Server-Level Roles**

http://aka.ms/fxwahr

public Server Role

All logins are automatically a member of the **public** server-level role, which is a special role that represents all server-level principals. The membership of the **public** role cannot be altered, and the **public** role cannot be removed.

The **public** role represents the default permissions on a server-level securable; when a server principal has no explicit permissions on an object, the permissions on the object held by the **public** role will be used.



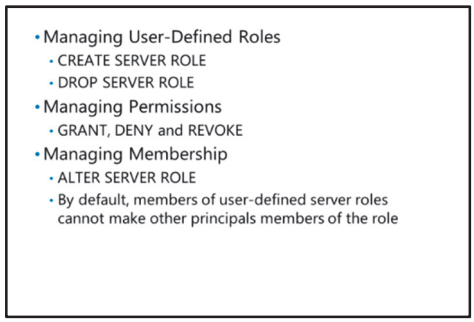
public Permissions

By default, the **public** role has CONNECT and VIEW ANY DATABASE permissions. Unlike other fixed server roles, you can grant additional permissions to **public**, but when doing so, you should be aware that you are effectively granting the permission to every login.

Best Practice: Controlling access to securable server objects by granting additional permissions to **public** is not recommended, because doing so might inadvertently grant more access to every login than is intended. For the same reason, you should regularly review the permissions granted to the **public** role.

Working with User-Defined Server Roles

When none of the fixed server-level roles exactly matches your security requirements, you can use user-defined server roles instead. You can create user-defined server roles by using Transact-SQL commands or through SQL Server Management Studio (SSMS), and then grant rights to your role. With user-defined roles, you can define the specific privileges required for your users and administrators, and limit the privileges to only those that are absolutely necessary.



You can create user-defined server-level roles by using the SSMS GUI or the CREATE SERVER ROLE statement, as shown in the following example:

Creating a User-Defined Server Role

```
CREATE SERVER ROLE app_admin;
```

When you create a server role, you might optionally define an owner for the role with the AUTHORIZATION clause. The role owner will have permission to add and remove members from the role. If you do not explicitly define an owner for the role, the role owner is the server principal executing the CREATE SERVER ROLE statement.

Unlike fixed server roles, the members of a user-defined server role do not, by default, have permission to add other security principals as members of the role.

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

CREATE SERVER ROLE (Transact-SQL)

http://aka.ms/tuuupj

You can remove a server role with the DROP SERVER ROLE statement.

Managing Permissions

You can grant a user-defined role permission to server-level objects with Transact-SQL using the GRANT, DENY and REVOKE commands, or through the SSMS GUI.

In the following example, the ALTER TRACE permission is granted to the **app_admin** role:

Granting Server Role Permissions

```
GRANT ALTER TRACE TO app_admin;
```

For more information on granting server-level permissions, see *GRANT (Transact-SQL)* in Microsoft Docs:

GRANT (Transact-SQL)

https://aka.ms/Ke2635

Managing Membership

You can add and remove logins to server-level roles by using SSMS or the ALTER SERVER ROLE Transact-SQL statement.

In the following code example, the **ADVENTUREWORKS\WebAdmins** login is added to the **app_admin** server-level role:

Adding a Login to a Server-Level Role

```
ALTER SERVER ROLE app_admin
ADD MEMBER [ADVENTUREWORKS\WebAdmins];
```

You can also create hierarchies of role membership by making a user-defined server role a member of another server role.

Best Practice: It is not recommended that you make user-defined server roles members of fixed server roles. Doing so will give control over membership of the fixed server role to members of the user-defined server role, which may lead to unintended escalation of privileges.

To remove a role member, use the ALTER SERVER ROLE statement with the DROP MEMBER clause.

For more information on the ALTER SERVER ROLE statement, see *ALTER SERVER ROLE (Transact-SQL)* in Microsoft Docs:

ALTER SERVER ROLE (Transact-SQL)

http://aka.ms/xutcov

Demonstration: Assigning Fixed and User-Defined Server Roles

In this demonstration, you will see how to work with:

- Server-scoped permissions.
- Fixed server roles.
- User-defined server roles.

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\Mod02** folder as Administrator. In the **User Account Control** dialog box, click **Yes**.
3. Start SQL Server Management Studio, and connect to the **MIA-SQL** database engine instance using Windows® authentication.
4. On the **File** menu, point to **Open**, and then click **Project/Solution**.
5. In the **Open Project** dialog box, navigate to **D:\Demofiles\Mod02\Project**, click **Project.ssmssln**, and then click **Open**.
6. In Solution Explorer, expand **Queries**, and then double-click **Demo 1 - server roles.sql**.
7. Execute the code under the heading for **Step 1** to show the permission hierarchy.
8. Execute the code under the heading for **Step 2** to create two logins.
9. Execute the code under the heading for **Step 3** to show that a new login is a member of **public**.
10. Execute the code under the heading for **Step 4** to demonstrate the permissions of a new login.
11. Execute the code under the heading for **Step 5** to add a login to the **diskadmin** role.
12. Execute the code under the heading for **Step 6** to verify the membership created in the previous step.
13. Execute the code under the heading for **Step 7** to create a user-defined server role.
14. Execute the code under the heading for **Step 8** to grant permissions to the new role.
15. Execute the code under the heading for **Step 9** to make a login a member of the new role.
16. Execute the code under the heading for **Step 10** to verify the membership created in the previous step.
17. Execute the code under the heading for **Step 11** to show the permissions of the login.
18. Execute the code under the heading for **Step 12** to remove the logins and role.
19. On the **File** menu, click **Close**.
20. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which fixed server role should be regarded as equivalent to sysadmin because of its ability to assign server principals to server roles?	
Select the correct answer.	
	serveradmin
	securityadmin
	processadmin
	setupadmin
	bulkadmin

Lesson 2
Working with Fixed Database Roles

In SQL Server, roles are supported at database level. The purpose of database-level roles is similar to the purpose of the server-level roles that you have already learned about in this module—to simplify the administration of database permissions. This lesson covers the fixed database roles available in every SQL Server database.

Lesson Objectives

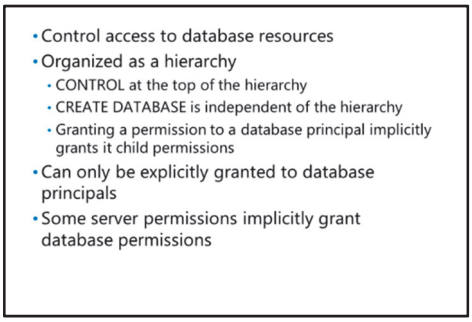
After completing this lesson, you will be able to:

- Explain database-scoped permissions.
- Use fixed database roles.
- Assign users to roles.
- Describe the purpose of the database owner role.

Database-Scoped Permissions

Permissions at database level relate to:

- Administrative actions related to the management of database objects (for example, BACKUP DATABASE, and commands to alter or create database objects).
- User actions on database objects (for example, CONNECT, SELECT, and UPDATE permissions).



You can view a complete list of database-scoped permissions using the system function **sys.fn_builtin_permissions**:

sys.fn_builtin_permissions Database

```
SELECT *
FROM sys.fn_builtin_permissions('DATABASE')
ORDER BY permission_name;
```

Like server permissions, database permissions are organized in a hierarchy. The hierarchy has two top-level nodes—CONTROL, which is the parent node for all other database permissions, and CREATE DATABASE, which has no children. Although database permissions can only be granted to database principals (users and roles), some server-level permissions implicitly grant database permissions.

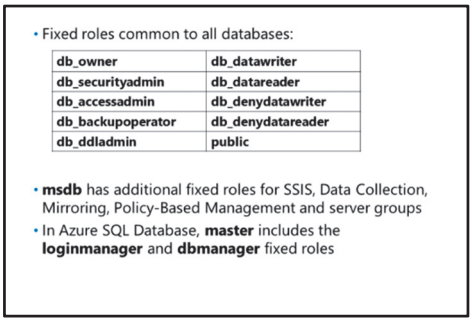
You can visualize the hierarchy of database permissions with the following query. The **parent_server_permission** column shows the server-level permission which implicitly grants the database permission:

Database Permissions Hierarchy

```
WITH recCTE
AS
(
      SELECT  permission_name, covering_permission_name AS parent_permission,
                  parent_covering_permission_name AS parent_server_permission,
                  1 AS hierarchy_level
      FROM sys.fn_builtin_permissions('DATABASE')
      WHERE covering_permission_name = ''

      UNION ALL

      SELECT  bp.permission_name, bp.covering_permission_name,
                  bp.parent_covering_permission_name AS parent_server_permission,
                  hierarchy_level + 1 AS hierarchy_level
      FROM recCTE AS r
      CROSS APPLY sys.fn_builtin_permissions('DATABASE') AS bp
      WHERE bp.covering_permission_name = r.permission_name
)
SELECT * FROM recCTE
ORDER BY hierarchy_level, permission_name;
```

Overview of Fixed Database Roles

Each database includes built-in fixed database-level roles with predefined rights for you to assign privileges for common scenarios. The following table lists the built-in fixed database-level roles:



Fixed Database-Level Role	Description
db_owner	Members of this role have comprehensive rights over the database, equivalent to the owner of the database. This includes the right to fully manage the database and also to drop the database, so you must use this role with caution.
db_securityadmin	Members of this role can grant permissions to configure database-wide settings and role membership.
db_accessadmin	Members of this role can manage access to the database by Windows logins, Windows groups, and SQL Server logins.
db_backupoperator	Members of this role can back up the database. Note that this role does not have the right to restore the database.

Fixed Database-Level Role	Description
db_ddladmin	Members of this role can run any Database Definition Language (DDL) Transact-SQL commands in the database. DDL is the portion of the Transact-SQL language that deals with creating, altering, and deleting database objects.
db_datawriter	Members of this role can change (INSERT, UPDATE, and DELETE) data in the database.
db_datareader	Members of this role can read data from all database tables.
db_denydatawriter	Members of this role cannot change (INSERT, UPDATE, and DELETE) data in the database.
db_denydatareader	Members of this role cannot read data from any database tables.
public	All database users belong to the **public** role. When a user has no explicit or inherited permissions on a database object, the permissions granted to **public** will be used.

To keep to the principle of least privilege, you should avoid using fixed database roles as far as possible. Unless a fixed database role holds exactly the permissions needed for a group of database principals, you should create a user-defined database role with only the permissions the principals require.

To view membership of database-level roles you can query the **sys.database_role_members** system view:

Database Role Membership

```
SELECT rdp.name AS role_name, rdm.name AS member_name
FROM sys.database_role_members AS rm
JOIN sys.database_principals AS rdp
ON rdp.principal_id = rm.role_principal_id
JOIN sys.database_principals AS rdm
ON rdm.principal_id = rm.member_principal_id
ORDER BY role_name, member_name;
```

msdb Fixed Database Roles

The **msdb** system database contains additional fixed database roles that relate to the administration of SSIS, Data Collection, Mirroring, Policy-Based Management, and server groups.

For more information on database-level roles, including **msdb** fixed database roles, see *Database-Level Roles* in Microsoft Docs:

Database-Level Roles

http://aka.ms/q4b8tn

master Database Roles in Azure SQL Database

Because you do not have administrative control of server-level objects in Azure SQL Database, the **master** database in an Azure SQL Database instance contains two fixed database roles that you can use to grant logins the permission to create new logins or new databases:

- **dbmanager**. Members of this role in the **master** database may create, alter, and drop databases on the Azure SQL Database instance.
- **loginmanager**. Members of this role in the **master** database may create, alter, and drop logins on the Azure SQL Database instance.

For more information on security configuration in Azure SQL Database, see *Controlling and granting database access* in the Microsoft Azure documentation:

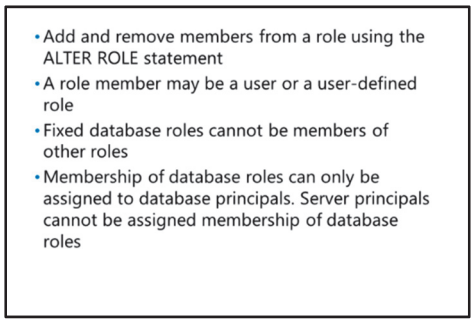
Controlling and granting database access

http://aka.ms/dq8nyh

Assigning Database Principals to Database Roles

You add and remove database principals from database roles using the ALTER ROLE command.

The following example demonstrates the ALTER ROLE command being used to add the user **amy0** to the fixed database role **db_backupoperator**, and then remove **amy0** from **db_backupoperator**:



ALTER ROLE Example

```
ALTER ROLE db_backupoperator ADD MEMBER
amy0;
GO
ALTER ROLE db_backupoperator DROP MEMBER
amy0;
GO
```

Note that:

- You may make a database user or a user-defined database role a member of a database role.
- You may not make a fixed database role a member of another database role.
- Membership of database roles is limited to database principals. Server principals cannot be made members of database roles.

For more information on the ALTER ROLE command, see *ALTER ROLE (Transact-SQL)* in Microsoft Docs:

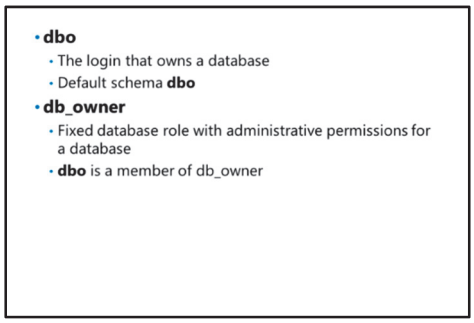
ALTER ROLE (Transact-SQL)

http://aka.ms/i5ddak

Database Owner

When discussing database ownership in SQL Server, there are two related concepts you must understand:

- The **dbo** user.
- The **db_owner** role.



The dbo User

The database user **dbo** is a built-in database user that acts as a database-level alias for the login which owns the database. Each database has only one owner; you may change the owner of a database using the ALTER DATABASE command.

The **dbo** user has a default schema in the database, also called **dbo**. Database objects created by members of the **sysadmin** fixed server role are added by default to the **dbo** schema.

The db_owner Role

The fixed database role **db_owner** is a role whose members have full administrative permissions for a database. The **dbo** user is, by default, a member of the **db_owner** role, but you may add as many members to the **db_owner** role as you wish. Amongst other permissions, members of the **db_owner** role have permission to create, delete, and alter database objects in any schema, including the **dbo** schema.

Demonstration: Managing Database Roles and Users

In this demonstration, you will see how to work with database roles and users.

Demonstration Steps

1. In SQL Server Management Studio, in Object Explorer, click **Connect**, and then click **Database Engine**.
2. In the **Connect to Server** dialog box, in the **Server name** box, type the name of your Azure instance running the **AdventureWorksLT** database, for example, **<servername>.database.windows.net**.
3. In the **Authentication** list, click **SQL Server Authentication**, in the **Login** box, type **Student**, in the **Password** box, type **Pa55w.rd**, and then click **Connect**.
4. In Solution Explorer, double-click **Demo 2 - database roles.sql**.
5. On the **Query** menu, point to **Connection**, and then click **Change Connection**.
6. In the **Connect to Database Engine** dialog box, in the **Server name** list, click **<servername>.database.windows.net**, in the **Password** box, type **Pa55w.rd**, and then click **Connect**.
7. Execute the code under the heading **Step 1** to create a new login.
8. On the toolbar, in the **Available Databases** list, click **AdventureWorksLT**.
9. Execute the code under the heading for **Step 2** to create a new user.
10. Execute the code under the heading for **Step 3** to demonstrate the database permissions hierarchy.
11. Execute the code under the heading for **Step 4** to demonstrate that the user is a member of the **public** database role by default.
12. Execute the code under the heading for **Step 5** to add the user to two fixed database roles.
13. Execute the code under the heading for **Step 6** to verify the user's role memberships.
14. Execute the code under **Step 7** to remove the demonstration objects.
15. On the **Query** menu, point to **Connection**, and then click **Change Connection**.
16. In the **Connect to Database Engine** dialog box, in the **Server name** list, click **<servername>.database.windows.net**, in the **Password** box, type **Pa55w.rd**, and then click **Connect**.
17. Execute the code under **Step 8**.
18. On the **File** menu, click **Close**.
19. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which of the following statements is incorrect (select one)?	
Select the correct answer.	
	dbo and db_owner are terms that refer to the same thing.
	dbo is a database user that is an alias for the login that owns the database.
	db_owner is a fixed database role with full administrative permissions over a database.
	The db_owner role can have more than one member.
	By default, dbo is a member of the db_owner role.

Lesson 3
User-Defined Database Roles

When you want finer-grained control over permissions on database objects than is offered by the fixed database roles, you can define your own database roles and grant them only the permissions they need. This lesson covers two types of user-defined database roles supported by SQL Server—database roles and application roles.

Lesson Objectives

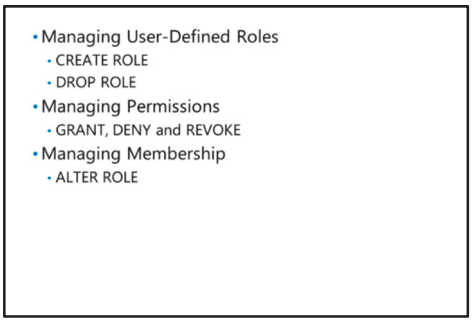
After completing this lesson, you will be able to:

- Create user-defined database roles.
- Describe typical scenarios where user-defined database roles might be used.
- Explain application roles.

Working with User-Defined Database Roles

Managing database-level roles is similar to managing server-level roles.

You can create user-defined database-level roles by using SSMS or the CREATE ROLE statement, as shown in the following example:



Creating a User-Defined Database Role

```
CREATE ROLE product_reader;
```

When you create a database role, you may optionally define an owner for the role with the AUTHORIZATION clause. The role owner will have permission to add and remove members from the role. If you do not explicitly define an owner for the role, the role owner is the database principal executing the CREATE ROLE statement. The owner of a role may be a database user, user-defined database role, or a fixed database role.

For more information on the CREATE ROLE statement, see *CREATE ROLE (Transact-SQL)* in Microsoft Docs:

CREATE ROLE (Transact-SQL)

http://aka.ms/ni4w6l

You can remove a database role with the DROP ROLE statement. You cannot remove a role that is the owner of one or more securable objects.

Managing Permissions

You may grant user-defined role permissions to database objects with Transact-SQL using the GRANT, DENY and REVOKE commands or through the SSMS GUI.

In the following example, the SELECT permission on the **Production.Product** table is granted to the **product_reader** role:

Database Role Grant

```
GRANT SELECT ON Production.Product TO product_reader;
```

For more information on commands used to manage database permissions, see *GRANT(Transact-SQL)* in Microsoft Docs:

GRANT (Transact-SQL)

https://aka.ms/Ke2635

Managing Membership

You can add and remove logins to database-level roles by using SSMS or the ALTER ROLE Transact-SQL statement.

In the following code example, the **WebApp** user is added to the **product_reader** role:

Adding a User to a Database-Level Role

```
ALTER ROLE product_reader
ADD MEMBER WebApp;
```

You may also create hierarchies of role membership by making a user-defined database role a member of another database role.

To remove a role member, use the ALTER ROLE statement with the DROP MEMBER clause.

For more information on the ALTER ROLE statement, see *ALTER ROLE (Transact-SQL*) in Microsoft Docs:

ALTER ROLE (Transact-SQL)

http://aka.ms/i5ddak

Applying Roles in Common Scenarios

In general, you use database roles where you want to restrict one or more users to the same set of permissions, following the principle of least privilege. This topic gives some examples of scenarios where you might use database roles.

- Controlling access to database objects
  - Object level
  - Schema level
  - Database level
- Controlling access to database-level actions
  - For example, SHOWPLAN
- As an abstraction layer
  - When details of users are not known—for example, during development

Note: You might consider creating a database role even when the role will initially have only one member, because you might want to grant the same permissions to other database principals at a later date.

Note: All the examples in this topic use a role called **example_role**. The code needed to create this role is not shown.

Controlling Access to Database Objects

You might want to restrict a group of users to SELECT permissions on a table, or EXECUTE permissions on a stored procedure. This is a common scenario for application or service accounts.

Database Role—Single Object Permissions

```
GRANT SELECT ON Production.Product TO example_role;
GO

GRANT EXECUTE ON dbo.uspGetOrderTrackingByTrackingNumber TO example_role;
GO
```

You might grant a role permission on a database schema; the permissions you grant at schema level automatically apply to all the objects in the schema:

Database Role – Schema Level Permissions

```
GRANT SELECT ON SCHEMA::Production TO example_role;
GO
```

You might grant a role permission at database level; the permissions you grant at database level automatically apply to all the objects in the database:

Database Role – Database Level Permissions

```
GRANT SELECT ON DATABASE::AdventureWorks TO example_role;
GO
```

 **Note:** This example is equivalent to the **db_datareader** fixed database role.

Controlling Access to Database-Level Actions

To grant members of a role permission to view query execution plans, grant the SHOWPLAN permission:

Database Role – SHOWPLAN

```
GRANT SHOWPLAN TO example_role;
GO
```

 **Note:** The SHOWPLAN permission allows a database principal to view the query execution plan for queries that the principal has permission to execute.
In a full SQL Server installation, to view the contents of the query plan cache, you must have the server-level VIEW SERVER STATE permission; VIEW SERVER STATE implicitly grants SHOWPLAN at database level.
In Azure SQL Database, there is no VIEW SERVER STATE permission. On premium performance tiers, the equivalent permission is VIEW DATABASE STATE. For other performance tiers, only the admin account may view the plan cache.

Roles As an Abstraction Layer

With database roles, you can verify your security configuration before you have details of all the users of a database. As users are added to the system, you can make them members of the relevant role.

This approach is useful when you want to add database permission scripts to a source control system during application development.

Demonstration: User-Defined Database Roles

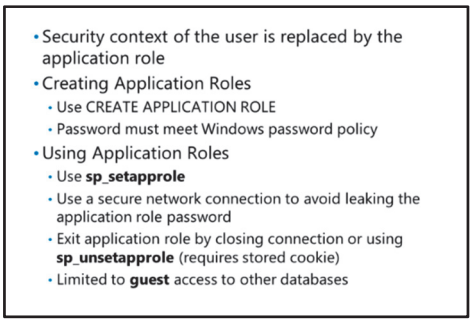
In this demonstration, you will see how to work with user-defined database roles.

Demonstration Steps

1. In SQL Server Management Studio, in Solution Explorer, double-click **Demo 3 - user database roles.sql**.
2. On the **Query** menu, point to **Connection**, and then click **Change Connection**.
3. In the **Connect to Database Engine** dialog box, in the **Server name** list, click **MIA-SQL**, in the **Authentication** box, click **Windows Authentication**, and then click **Connect**.
4. Execute the code under the heading **Step 1** to create a user-defined database role.
5. Execute the code under the heading **Step 2** to grant SELECT permissions on the **HumanResources** schema to the role.
6. Execute the code under the heading **Step 3** to verify the role permissions.
7. Execute the code under the heading **Step 4** to add two users to the role.
8. Execute the code under the heading **Step 5** to verify the role's membership.
9. Execute the code under the heading **Step 6** to remove the demonstration role.
10. On the **File** menu, click **Close**.
11. Keep SQL Server Management Studio open for the next demonstration.

Defining Application Roles

An application role is a database-level role that makes it possible for a set of permissions to be associated with an application, rather than a user or group of users. Instead of granting the permissions required by an application to users—either directly, or through a database role—the permissions are granted to the application role, available to any user of the database.



Unlike database roles, application roles have no members; instead, application roles are secured with a password. Any user with access to a database (through the **public** role) may use an application role when they provide the application role password. When a user activates an application role—using the **sp_setapprole** system stored procedure—the security context of the user's session switches to that of the application role.

Note: An application role completely replaces a user's security context. None of the user's permissions apply when they activate an application role.

You might use an application role if you want to use Windows authentication to give users access to a database, but do not want to grant them the same permissions as the applications they use. For example, members of a Windows group use an application that requires full access to tables in the **Purchasing** schema. You want the users to be able to access these tables through the application, but do not want them to be able to execute impromptu SELECT, INSERT and UPDATE statements. An application role provides one method of achieving this.

For more information about application roles, see *Application Roles* in Microsoft Docs:

Application Roles

http://aka.ms/xoywhz

Creating Application Roles

When you create an application role, you must specify a password. The password must meet the Windows password policy requirements of the computer running the SQL Server instance.

You create application roles using the CREATE APPLICATION ROLE statement:

CREATE APPLICATION ROLE

```
CREATE APPLICATION ROLE app_role_1 WITH PASSWORD = N'DPwp4O3zl9M2YS77yAjD';
```

For more information on the CREATE APPLICATION ROLE statement, see *CREATE APPLICATION ROLE (Transact-SQL)* in Microsoft Docs:

CREATE APPLICATION ROLE (Transact-SQL)

http://aka.ms/b4agp9

You can drop application roles with the DROP APPLICATION ROLE statement.

Using Application Roles

After you have created an application role and assigned it the required permissions, it can be activated by executing the **sp_setapprole** system stored procedure.

The following code example shows how to activate an application role:

Activating an Application Role

```
EXEC sp_setapprole @rolename = N'app_role_1', @password = N'DPwp4O3zl9M2YS77yAjD';
```

Note: If calls to the **sp_setapprole** stored procedure will be made across a network, you must ensure that the connection is encrypted—for example, using SSL or IPSec—to avoid exposing the application role password, because the password is effectively sent in plain text. **sp_setapprole** accepts an optional @encrypt = 'odbc' parameter, but this uses an ODBC function that obfuscates the password rather than truly encrypting it. The ODBC encrypt function is not supported by the **SqlClient** library.

An application role remains active until the user disconnects from SQL Server or it is deactivated by using the **sp_unsetapprole** stored procedure. However, to use **sp_unsetapprole**, you must specify a cookie that was generated when the application role was activated.

The following code example shows how to create a cookie when activating an application role, and how to use the cookie to deactivate the application role when it is no longer required:

Activating and Deactivating an Application Role

```
DECLARE @cookie_store varbinary(8000);
EXEC sp_setapprole N'app_role_1', N'DPwp4O3zl9M2YS77yAjD', @fCreateCookie = true, @cookie
= @cookie_store OUTPUT;

-- Perform the operation that requires application role permissions

EXEC sp_unsetapprole @cookie_store;
```

 **Note:** If they do not store application role cookies and revert to the connection security context using **sp_unsetapprole**, applications using an application role must disable connection pooling.

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

 **sp_setapprole (Transact-SQL)**

http://aka.ms/homihf

Application roles are database principals that are not linked to a server principal. As a result, application roles are restricted to the permissions granted to the **guest** user in other databases on the instance. In databases where the **guest** user is disabled (the default behavior), an application role has no access to the database at all.

Demonstration: Application Roles

In this demonstration, you will see how to work with application roles.

Demonstration Steps

1. In SQL Server Management Studio, in Solution Explorer, double-click **Demo 4 - application roles.sql**.
2. On the **Query** menu, point to **Connection**, and then click **Change Connection**.
3. In the **Connect to Database Engine** dialog box, in the **Server name** list, click **MIA-SQL**, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
4. Execute the code under the heading **Step 1** to create an application role.
5. Execute the code under the heading **Step 2** to grant permissions to the role.
6. Execute the code under the heading **Step 3** to demonstrate behavior before the application role is activated.
7. Execute the code under the heading **Step 4** to activate the application role.
8. Execute the code under the heading **Step 5** to demonstrate that the application role permissions apply.
9. Execute the code under the heading **Step 6** to show how the user's identity is represented.
10. Execute the code under the heading **Step 7** to show that cross-database access is limited.
11. Execute the code under the heading **Step 8** to exit the application role.

12. Execute the code under the heading **Step 9** to show how the user's identity is represented.
13. Execute the code under the heading **Step 10** to remove the application role.
14. On the **File** menu, click **Close**.
15. Close SSMS without saving any changes.

Sequencing Activity

To indicate the correct order, number each of the following steps, which describe the sequence of actions when an application uses an application role.

	Steps
	An administrator makes the user a member of the database public role.
	The user starts an application, which connects to the database using the user's credentials.
	The application activates an application role, using the password held in the application's configuration file.
	The user works in the application with the security context of the application role.
	The user closes the application.

Lab: Assigning Server and Database Roles

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. Following an internal security audit, the company aims to simplify the administration of database security by making security controls more consistent. You are a database administrator for Adventure Works, tasked with implementing the new security controls through server-level and database-level roles.

Objectives

After completing this lab, you will be able to:

- Implement and manage fixed server roles and user-defined server roles.
- Implement and manage fixed database roles.
- Implement and manage user-defined database roles, including application roles.
- Verify role-based security settings.

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Assigning Server Roles

Scenario

The **MIA-SQL** SQL Server instance will be used to host application databases, including the **salesapp1** database. It is a requirement that all corporate IT support personnel can alter any login and view any database on all SQL Server instances. They should have no additional administrative privileges on the SQL Server instance.

The **adventureworks.msft** domain includes the following global group relevant to this task:

ADVENTUREWORKS\IT_Support: Contains all IT support personnel.

The domain administrator has created the following domain local group, with the members shown:

- **ADVENTUREWORKS\Database_Managers:**
    - **ADVENTUREWORKS\IT_Support**

A server login has already been created for the **ADVENTUREWORKS\Database_Managers** on the **MIA-SQL** instance, but no permissions have been assigned.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create a Server Role

3. Assign Server-Level Permissions

4. Assign Role Membership

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab02\Starter** folder as Administrator.

▶ Task 2: Create a Server Role

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of SQL Server.
2. Open the project file **D:\Labfiles\Lab02\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - server roles.sql**.
3. Under the heading for **Task 1**, write a script to create a server role called **database_manager**.

▶ Task 3: Assign Server-Level Permissions

- Under the heading for **Task 2**, edit the query to grant permissions to the **database_manager** role. The role needs permissions to:
  - Alter any login
  - View any database

▶ Task 4: Assign Role Membership

1. Under the heading for **Task 3**, write a query to make the **ADVENTUREWORKS\Database_Managers** login a member of the **database_managers** role. Note that a login already exists for **ADVENTUREWORKS\Database_Managers**, so you do not have to create one.
2. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have created the **database_manager** server role, granted permissions to members to alter any login and alter any database, and granted membership to the members of the **Database_Managers** login.

Exercise 2: Assigning Fixed Database Roles

Scenario

In addition to the server-level permissions you assigned to the **ADVENTUREWORKS\Database_Managers** login as part of the previous exercise, members of **ADVENTUREWORKS\Database_Managers** should be able to add and remove users from the **salesapp1** database, and administer backups of the **salesapp1** database.

The main tasks for this exercise are as follows:

1. Create a Database User and Assign Fixed Database Roles

▶ Task 1: Create a Database User and Assign Fixed Database Roles

1. Using the SSMS GUI, create a user in the **salesapp1** database for the **ADVENTUREWORKS\Database_Managers** login.
2. Grant the user membership of the following roles:
   - **db_accessadmin**
   - **db_backupoperator**
3. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have mapped the **Database_Managers** login to the **salesapp1** database and added them to the **db_backupoperator** and **db_accessadmin** roles.

Exercise 3: Assigning User-Defined Database Roles

Scenario

The users of the **salesapp1** database belong to two Active Directory groups:

- **ADVENTUREWORKS\InternetSales_Users**
- **ADVENTUREWORKS\InternetSales_Managers**

Both of these groups already have a login to the **MIA-SQL** instance, but are not configured to access the **salesapp1** database.

You must grant the following permissions:

- Members of **ADVENTUREWORKS\InternetSales_Users** should have permission to:
  - SELECT from all tables in the **Sales** schema.
- Members of **ADVENTUREWORKS\InternetSales_Managers** should have permission to:
  - SELECT from all tables in the **Production** and **Sales** schemas.
  - UPDATE the **Sales.Orders** table and the **Sales.OrderDetails** table.

The main tasks for this exercise are as follows:

1. Create a Database Principal in SSMS
2. Create a User-Defined Database Role in SSMS
3. Create a Database Principal by Using Transact-SQL
4. Create User-Defined Database Roles by Using Transact-SQL
5. Grant Permissions to User-Defined Database Roles Using Transact-SQL
6. Add a Database Principal to a Role Using Transact-SQL

▶ Task 1: Create a Database Principal in SSMS

- Using the SSMS GUI, create the following user in the **salesapp1** database:
  - **internetsales_user**, mapped to the **ADVENTUREWORKS\InternetSales_Users** login.

▶ Task 2: Create a User-Defined Database Role in SSMS

1. Using the SSMS GUI, create a role called **sales_reader**, with SELECT permission on the **Sales** schema.
2. Make the **internetsales_user** user a member of the role.

▶ Task 3: Create a Database Principal by Using Transact-SQL

1. In SSMS Object Explorer, open the query file **Lab Exercise 03 - database roles.sql**.
2. Execute the code under the heading for **Task 3** to create the **internetsales_manager** user in the **salesapp1** database.

▶ Task 4: Create User-Defined Database Roles by Using Transact-SQL

- Under the heading for **Task 4**, write a query to create two user-defined database roles, called:
  - **production_reader**
  - **sales_order_writer**

▶ Task 5: Grant Permissions to User-Defined Database Roles Using Transact-SQL

1. Under the heading for **Task 5**, write and execute a query to grant SELECT permissions on the **Production** schema to the **production_reader** role.
2. Write and execute a query to grant UPDATE permissions on the **Sales.Orders** table and the **Sales.OrderDetails** table to the **sales_order_writer** role.

▶ Task 6: Add a Database Principal to a Role Using Transact-SQL

1. Under the heading for **Task 6**, edit the query to make the **internetsales_manager** user a member of the **sales_reader**, **sales_order_writer**, and **production_reader** roles.
2. Execute the query.
3. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have created user-defined database roles and assigned them to database principals.

Exercise 4: Verifying Security

Scenario

You have created the required server and database principals, and applied appropriate permissions. Now you must verify that the permissions give users access to the data they require, but do not permit any unnecessary data access.

All the command prompt commands used in this exercise can be found in:
D:\Labfiles\Lab02\Solution\Project\Project\Lab Exercise 04 - verify.txt.

The main tasks for this exercise are as follows:

1. Test IT Support Permissions
2. Test Sales Employee Permissions
3. Test Sales Manager Permissions

▶ Task 1: Test IT Support Permissions

1. Open a command prompt and enter the following command (which opens the **sqlcmd** utility as **ADVENTUREWORKS\AnthonyFrizzell**):

```
runas /user:adventureworks\anthonyfrizzell /noprofile sqlcmd
```

2. When you are prompted for a password, enter **Pa55w.rd**.
3. In the SQLCMD window, enter the following commands to verify your identity:

```
SELECT SUSER_NAME();
GO
```

Note that SQL Server identifies Windows group logins using their individual user account, even though there is no individual login for that user. **ADVENTUREWORKS\AnthonyFrizzell** is a member of the **ADVENTUREWORKS\IT_Support** global group, which is in turn a member of the **ADVENTUREWORKS\Database_Managers** domain group.

4. In the SQLCMD window, execute an ALTER LOGIN to change the password of the login for the **Marketing_Application** login. Your code should look similar to this:

```
ALTER LOGIN login_name WITH PASSWORD = 'NewPa55w.rd';
GO
```

5. In the SQLCMD window, enter an ALTER LOGIN command to disable the **ADVENTUREWORKS\WebApplicationSvc** login. Your command should look like this:

```
ALTER LOGIN login_name DISABLE;
GO
```

6. Close the SQLCMD window.
7. In SSMS, view the properties of the **ADVENTUREWORKS\WebApplicationSvc** login, verify that the login is disabled, and then re-enable it.

▶ Task 2: Test Sales Employee Permissions

1. In a command prompt window, enter the following command to run **sqlcmd** as **ADVENTUREWORKS\DanDrayton**. This user is a member of the **ADVENTUREWORKS\Sales_NorthAmerica** global group, which is in turn a member of the **ADVENTUREWORKS\InternetSales_Users** domain group:

```
runas /user:adventureworks\dandrayton /noprofile sqlcmd
```

2. When you are prompted for a password, enter **Pa55w.rd**.
3. Verify that you can query the **Sales.Orders** table in the **salesapp1** database. For example, execute the following query:

```
SELECT TOP (10) * FROM salesapp1.Sales.Orders;
GO
```

4. Verify that you cannot update the **Sales.Orders** table in the **salesapp1** database. For example, execute the following query:

```
UPDATE salesapp1.Sales.Orders SET shippeddate = getdate() WHERE orderid = 10257;
GO
```

   This command should return an error if the role permissions are correctly applied.

5. Close the SQLCMD window.

▶ Task 3: Test Sales Manager Permissions

1. In a command prompt window, enter the following command to run **sqlcmd** as **ADVENTUREWORKS\DeannaBall**. This user is a member of the **ADVENTUREWORKS\Sales_Managers** global group, which is in turn a member of the **ADVENTUREWORKS\InternetSales_Managers** domain group:

```
runas /user:adventureworks\deannaball /noprofile sqlcmd
```

2. When you are prompted for a password, type **Pa55w.rd**.
3. Verify that you can query the **Sales.Orders** table in the **salesapp1** database. For example, execute the following query:

```
SELECT TOP (10) * FROM salesapp1.Sales.Orders;
GO
```

4. Verify that you can query the **Production.Suppliers** table in the **salesapp1** database. For example, execute the following query:

```
SELECT TOP (10) * FROM salesapp1.Production.Suppliers;
GO
```

5. Verify that you can update the **Sales.Orders** table in the **salesapp1** database. For example, execute the following query:

```
UPDATE salesapp1.Sales.Orders SET shippeddate = getdate() WHERE orderid = 10257;
GO
```

6. Close the SQLCMD window.
7. Close the Command Prompt window.
8. Close SQL Server Management Studio without saving any changes.

Results: At the end of this exercise, you will have verified your new security settings.

Question:

Your organization wants to track data access by individual Windows users. Does this mean you cannot base logins on Windows groups?

Module Review and Takeaways

In this module, you have learned how to implement role-based security in a SQL Server Database Engine instance.

Best Practice: When implementing role-based security in SQL Server, consider the following best practices:

- Use Windows group logins linked to roles to simplify ongoing management where possible.
- Aim to grant the minimum number of explicit permissions possible to meet the security requirements, and use membership of roles and inheritance to ensure the correct effective permissions.
- Ensure every database user has only the permission they actually require.

Review Question(s)

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or False? When **sp_setapprole** is called with the @encrypt = 'odbc' parameter, the application role password is encrypted with strong encryption.	

Check Your Knowledge

Question	
Which permission is required to view the contents of the query plan cache for an on-premises SQL Server instance?	
Select the correct answer.	
	SHOWPLAN (database level)
	SHOWPLAN (server level)
	VIEW SERVER STATE (server level)
	VIEW SERVER STATE (database level)

Module 3

Authorizing Users to Access Resources

Contents:

Module Overview

In the previous modules, you have seen how Microsoft® SQL Server® security is organized and how sets of permissions can be assigned at the server and database level by using fixed server roles, user-defined server roles, fixed database roles, and application roles. The final step in authorizing users to access SQL Server resources is the authorization of users and roles to access server and database objects.

In this module, you will see how these object permissions are managed. In addition to access permissions on database objects, SQL Server provides the ability to determine which users are allowed to execute code, such as stored procedures and functions. In many cases, these permissions and the permissions on the database objects are best configured at the schema level rather than at the level of the individual object. Schema-based permission grants can simplify your security architecture. You will explore the granting of permissions at the schema level in the final lesson of this module.

Objectives

After completing this module, you will be able to:

- Authorize user access to objects.
- Authorize users to execute code.
- Configure permissions at the schema level.

Lesson 1
Authorizing User Access to Objects

Before moving on to managing permissions on code, you need to consider how permissions are managed on database objects. SQL Server has a fine-grained security model that means you can grant the minimum permissions to users that will allow them to do their work. In particular, permissions can be granted at the column level, not just at the table and view level. You will also see how you can delegate the work of granting permissions to other users.

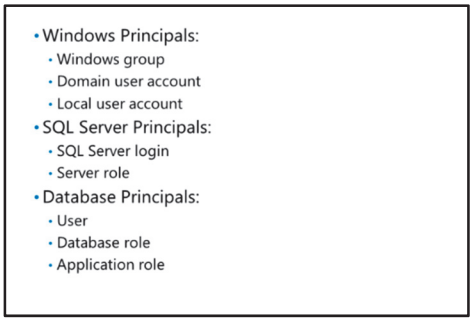
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the role of principals.
- Explain the role of securables.
- Use the GRANT, REVOKE, and DENY commands.
- Secure tables and views.
- Implement column-level security.
- Implement row-level security.
- Delegate the ability to assign permissions.

What Are Principals?

Principals are entities that can request and be granted access to SQL Server resources. Like other components in the SQL Server authorization model, principals are arranged in a hierarchy.

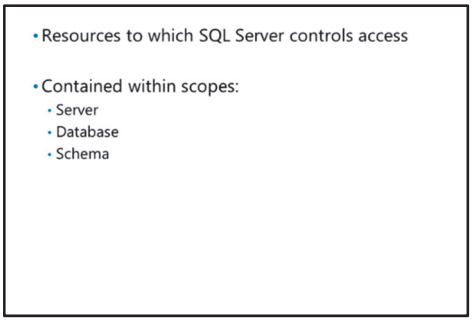


- At the Windows® level, principals include users and groups. If the server is part of a Windows domain, the users and groups can be domain-based and, regardless of domain membership, can be local server accounts.
- At the SQL Server level, you can create logins for users that are either not Windows users or users who are part of nontrusted Windows environments. For example, users in other Windows domains where no trust relationship exists with the domain containing the SQL Server system. Also, at the SQL Server level, fixed and user-defined server roles are a form of principal that contains other principals.
- At the database level, principals include database users, fixed and user-defined database roles, and application roles.

Every principal has two numeric IDs associated with it—a principal ID and a security identifier (SID).

What Are Securables?

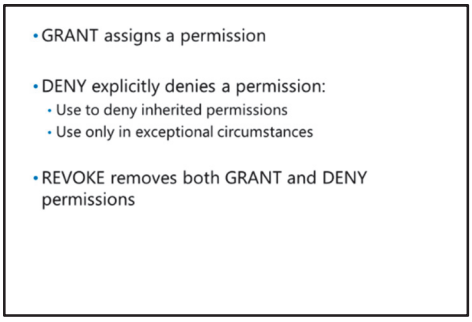
Securables are the resources to which the SQL Server Database Engine authorization system regulates access. Some securables can be contained within others, creating nested hierarchies called scopes that can themselves be secured. The securable scopes are server, database, and schema. It is important to understand the different securable scopes in SQL Server to effectively plan your security model.



SQL Server principals are assigned or denied access to specific securables so that they can complete their work.

GRANT, REVOKE, DENY

Permissions are managed by using the GRANT, DENY, and REVOKE Transact-SQL statements. Most permissions (but not all) can also be managed by using SQL Server Management Studio (SSMS).



GRANT and REVOKE

A user who has not been granted permission cannot perform the action related to it. For example, users have no permission to SELECT data from tables if they have not been granted permission at some level. Some other database engines consider this to be an implicit form of denial. You can use the GRANT statement to assign permissions on a securable to database users. Similarly, you can use the REVOKE statement to remove the same permissions.

DENY

In ANSI (American National Standards Institute) SQL, if a user does not have permission to perform an action, they cannot do so. Therefore, ANSI SQL does not provide a DENY statement.

Because SQL Server is closely linked with the Windows operating system and its group membership system, a Windows user can receive permissions through their group membership. Therefore, there may be cases where a user has a permission granted through their group membership that you might want to remove for the individual.

To support this scenario, SQL Server provides the DENY statement, which you can use to explicitly deny a user a permission that they may receive from membership of a group or role. This is very similar to the process for denying permissions work in Windows. For a Windows example, consider that you could decide that all members of the Salespeople group can access a color printer—except Holly (who is a member of Salespeople), because she causes problems with it. You grant access to the Salespeople group then deny it to Holly. SQL Server works in the same way. You could grant SELECT permission on a table to every member of the Salespeople role, and then deny Holly access to that table.

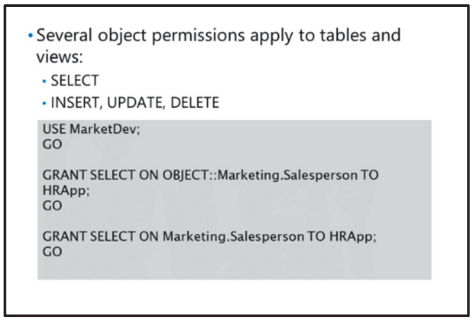
As with Windows, you should use DENY sparingly. If you need to DENY many permissions, it might indicate a potential problem with your security design.

Note: The REVOKE statement revokes both GRANT and DENY statements.

Securing Tables and Views

The permissions to access data that apply to tables and views are SELECT, INSERT, UPDATE, DELETE, and REFERENCES.

In the following example, SELECT permission on the **Marketing.Salesperson** object (which is likely to be a table or a view) is being granted to the **HRApp** user in the **MarketDev** database:



Applying Permissions

```
USE MarketDev;
GO

GRANT SELECT ON OBJECT::Marketing.Salesperson TO HRApp;
GO

GRANT SELECT ON Marketing.Salesperson TO HRApp;
GO
```

Note that two forms of the command are shown. While the full terminology involves OBJECT:: as a prefix, this is optional. In the second example, the same GRANT statement is shown without the OBJECT:: prefix.

It is not strictly necessary to specify the schema for the table or view, but doing so is highly recommended to ensure that permissions are granted on the intended object. If the schema name is not specified, the default schema for the user granting the permission is used. If the object is not found in the user's default schema, the **dbo** schema is used instead.

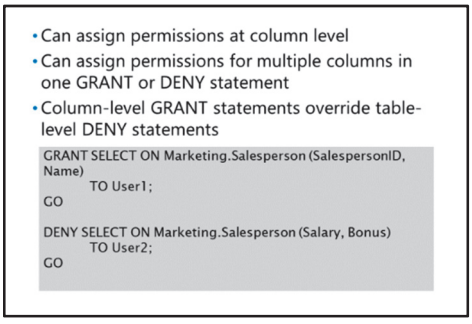
REFERENCES

While the meaning of the SELECT, INSERT, UPDATE, and DELETE permissions will likely be obvious to you, the meaning and purpose of the REFERENCES permission might not. You need to use the REFERENCES permission before a foreign key relationship can specify the object as a target, and is only required if no other permissions exist on the referenced object.

Column-Level Security

In addition to assigning permissions at table or view level, you can also allocate column-level permissions. This provides a more granular level of security for data in your database.

You do not need to execute separate GRANT or DENY statements for every column where you wish to assign permissions. When a set of columns needs to be controlled in the same way, you can provide a list of columns in a single GRANT or DENY statement.



Using the GRANT and DENY Statements for Columns

```
GRANT SELECT ON Marketing.Salesperson (SalespersonID, Name) TO User1;
GO

DENY SELECT ON Marketing.Salesperson (Salary, Bonus) TO User2;
GO
```

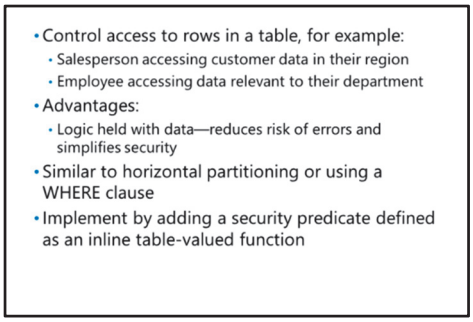
User1 can now access two columns in the Marketing.Salesperson table, but not the whole table.

Table-Level DENY and Column-Level GRANT

If you execute a DENY statement for a user at table level, and then execute a GRANT statement at column level, the user can still access the columns to which you grant access. However, if you then execute the table-level DENY statement again, the user is denied all permissions on the table, including on the columns to which they previously had access.

Row-Level Security

Row-level security (RLS) helps you to control access to rows in a table. This can be useful in a variety of scenarios, including when you want a salesperson to only access customer data for customers in their region; or when you want to limit an employee to only access data relevant to their department. One key advantage of this is that the logic for the separation is in the actual database. This reduces the risk of errors in the application that is allowing inappropriate access, and simplifies the security implementation for that table.



Functionally, this is similar to horizontal partitioning of data or including a WHERE clause in a query. You implement RLS by adding a security predicate defined as an inline table-valued function that returns 1 when the predicate is met.

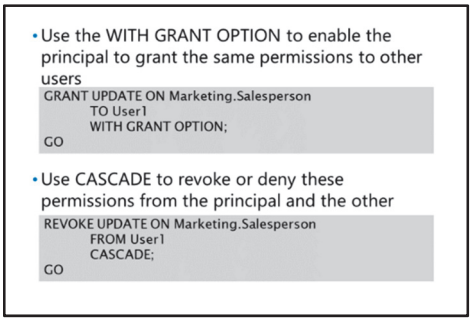
For more information about RLS, see *Row-Level Security* in Microsoft Docs:

Row-Level Security

http://aka.ms/Br9x2d

WITH GRANT Option

When you grant permissions to a principal, you can also give them the right to regrant the same permissions to other principals by using the WITH GRANT OPTION clause. This means you can delegate the responsibility for managing permissions, but you should use this with caution—because you then lose control of the security of that securable.



In the following example, User1 is given permission to perform updates on the **Marketing.Salesperson** table, in addition to the right to grant this same permission to other users:

Using the WITH GRANT OPTION Clause

```
GRANT UPDATE ON Marketing.Salesperson TO User1 WITH GRANT OPTION;
GO
```

CASCADE

The challenge of the WITH GRANT OPTION clause comes when you need to REVOKE or DENY the permission that you granted to James using the WITH GRANT OPTION. You do not know which other users James has already granted the permission to.

When revoking or denying a permission, you can use the CASCADE clause to also revoke or deny permissions from any users who had been granted them by User1.

Using the CASCADE Clause

```
REVOKE UPDATE ON Marketing.Salesperson FROM User1 CASCADE;
GO
```

In this example, the REVOKE statement will fail if you omit the CASCADE clause, because the GRANT statement included the WITH GRANT OPTION clause.

Demonstration: Authorizing User Access to Objects

In this demonstration, you will see how to view principals and grant permissions on database objects.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and MIA**20764C-MIA-SQL** virtual machines are running, and log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod03** folder as Administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. On the taskbar, click the **Microsoft SQL Server Management Studio**.
5. In the **Connect to Server** dialog box, click **Connect**.
6. On the **File** menu, point to **Open**, and then click **File**.

7. In the **Open File** dialog box, navigate to **D:\Demofiles\Mod03**, click **AuthorizingUserAccessToObjects.sql**, and then click **Open**.
8. Execute the code under the heading for **Step 1** to create a user for the demonstration.
9. Execute the code under the heading for **Step 2** to query the list of server principals. Note **Mod03Login** at the end of the list.
10. Execute the code under the heading for **Step 3** to query the list of database principals. Again, note **Mod03Login** in the list.
11. Execute the code under the heading for **Step 4** to grant SELECT permissions on the **Product** table to **Mod03Login**.
12. Execute the code under the heading for **Step 5** to change the execution context.
13. Execute the code under the heading for **Step 6** to test the permissions. Note that you can select from the **Product** table that you were granted permissions on, but not from the **ProductInventory** table.
14. Execute the code under the heading for **Step 7** to revert the execution context.
15. Execute the code under the heading for **Step 8** to grant SELECT permissions on specific columns in the **ProductInventory** table to **Mod03Login**.
16. Execute the code under the heading for **Step 9** to change the execution context.
17. Execute the code under the heading for **Step 10** to test the permissions. Note that the first query to select the two specific columns executes, but you cannot select all the columns from the **ProductInventory** table.
18. Execute the code under the heading for **Step 11** to revert the execution context.
19. On the **File** menu, click **Close**.
20. Leave SQL Server Management Studio open for the next demonstration.

    Question: What is the REFERENCES permission used for?

Lesson 2
Authorizing Users to Execute Code

In addition to providing you with control over who accesses data in your database or the objects in your server, SQL Server helps you to control which users can execute your code. Appropriate security control of code execution is an important aspect of your security architecture.

In this lesson, you will see how to manage the security of stored procedures and functions. You will also learn how to manage security for code that lives in .NET-managed code assemblies that are used with SQL CLR integration. Finally, you will see how ownership chains affect the security relationship between code and database objects.

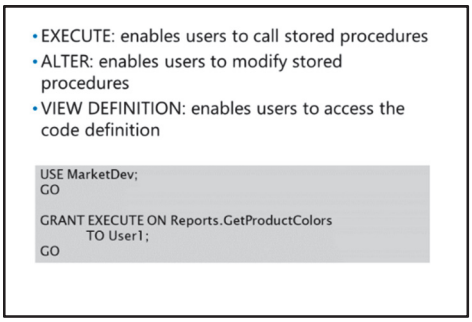
Lesson Objectives

After completing this lesson, you will be able to:

- Secure stored procedures.
- Secure user-defined functions.
- Secure managed code.
- Manage ownership chains.

Securing Stored Procedures

By default, users cannot execute stored procedures that other users create unless you grant them the EXECUTE permission on the stored procedure. In addition, they might also need permissions to access the objects that the stored procedure uses. You will discover more about this issue later in this lesson.



In the following example, User1 is granted EXECUTE permission on the **Reports.GetProductColors** stored procedure:

Granting EXECUTE Permissions on Stored Procedures

```
USE MarketDev;
GO

GRANT EXECUTE ON Reports.GetProductColors TO User1;
GO
```

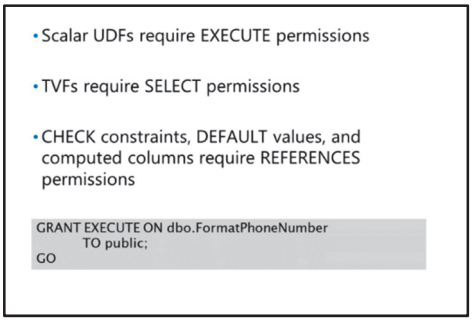
Managing Stored Procedures

There are two other permissions related to the management of stored procedures:

- The ALTER permission enables a user to change the definition of a stored procedure.
- The VIEW DEFINITION permission enables a user to view the code definition of the stored procedure.

Note: You can use SSMS or Transact-SQL to grant permissions on user stored procedures, but you can only grant permissions on system stored procedures by using Transact-SQL code.

Securing User-Defined Functions



You also need to assign users permissions to execute user-defined functions (UDFs). The permissions that you need to assign depend on the type of UDF you are working with.

- Scalar UDFs return a single value. Users accessing these functions require EXECUTE permission on the UDF.
- Table-valued UDFs (TVFs) return a table of results rather than a single value. Accessing a TVF requires SELECT permission, rather than EXECUTE permission, similar to the permissions on a table.
- It is uncommon to directly update a TVF. It is possible, however, to assign INSERT, UPDATE, and DELETE permissions on one form of TVF known as an inline TVF—in some cases, this particular form can be updated.

In addition to these permissions, there are scenarios where you also need to assign the REFERENCES permission to users so that they can correctly execute a UDF. These scenarios include functions that:

- Are used in CHECK constraints.
- Calculate values for DEFAULT constraints.
- Calculate values for computed columns.

Functions often provide low-risk, basic capabilities within systems. Because of this, it is fairly common practice to grant permissions on basic functions that are contained in a database to the **public** role of the database. This means that any user in the database can use those functions without needing further permission grants.

The following example shows how to grant execute permissions to the **public** role on the **dbo.FormatPhoneNumber** function:

Granting EXECUTE Permissions on UDFs

```
GRANT EXECUTE ON dbo.FormatPhoneNumber TO public;
GO
```

Note: Assigning permissions to the **public** role on stored procedures is not appropriate.

Securing Managed Code

Managed code is .NET Framework code that ships in assemblies. Assemblies can exist as DLL or EXE files; however, you can only load assemblies in DLL files in SQL Server by using SQL Server CLR integration. After you load an assembly, the procedures, functions, and other managed code objects appear as standard objects in SQL Server, and the standard object permissions apply. For example, users require EXECUTE permissions to run a stored procedure, whether it originates in an assembly or in Transact-SQL code.

- Additional permission requirements above those required for Transact-SQL code
- Assemblies are registered with one of three permissions sets:
  - SAFE (the default)
  - EXTERNAL_ACCESS
  - UNSAFE
- EXTERNAL_ACCESS and UNSAFE permission sets require additional configuration on the database

Permission Sets

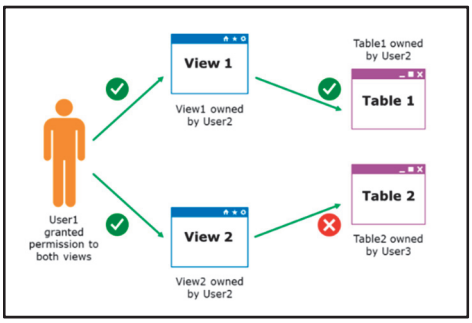
No matter what .NET Framework code is included in an assembly, the actions the code can execute are determined by the permission set specified when creating the assembly.

- The SAFE permission set strictly limits the actions that the assembly can perform and inhibits it from accessing external system resources. Code using this permission set can access the local instance of SQL Server by using a direct access path, called a context connection. The SAFE permission set is the default.
- The EXTERNAL_ACCESS permission set allows the code to access local and network resources, environment variables, and the registry. EXTERNAL_ACCESS is even necessary for accessing the same SQL Server instance if a connection is made through a network interface.
- The UNSAFE permission set relaxes many standard controls over code so you should avoid using it.

The EXTERNAL_ACCESS and UNSAFE permission sets require additional setup. You cannot specify the requirement for an EXTERNAL_ACCESS permission set when executing the CREATE ASSEMBLY statement. You should flag the database as TRUSTWORTHY (which is easy, but not recommended) or create an asymmetric key from the assembly file in the master database, create a login that maps to the key, and grant the login EXTERNAL ACCESS ASSEMBLY permission on the assembly.

Managing Ownership Chains

All database objects have owners. By default, the **principal_id(owner)** property is set to NULL for new objects and the owner of a schema automatically owns schema-scoped objects. The best practice is to have all objects owned by the schema object owner—therefore, an object with a NULL **principal_id** property inherits its ownership from the schema where it is contained.



When an object such as a stored procedure references another object, an ownership chain is established. An unbroken ownership chain exists when each object in the chain has the same owner. When an unbroken ownership chain exists, access is permitted to the underlying objects, and access is given to the top level objects.

Having the same owner for all objects in a schema (which itself also has an owner) simplifies permission management. However, it is still important to understand that ownership chain problems can occur and you should know how to resolve them.

Ownership chaining applies to stored procedures, views, and functions. The slide shows an example of how ownership chaining applies to views or stored procedures.

1. User1 has no permissions on the table owned by User2.
2. User2 creates a view that accesses the table and grants User1 permission to access the view. Access is granted as User2 is the owner of both the top level object (the view) and the underlying object (the table).
3. User2 then creates a view that accesses a table owned by User3. Even if User2 has permission to access the table and grants User1 permission to use the view, User1 will be denied access because of the broken chain of ownership from the top level object (the view) to the underlying object (the table).
4. However, if User3 grants User1 permissions directly on the underlying table, he can then access the view that User2 created to access that table.

Demonstration: Authorizing Users to Execute Code

In this demonstration, you will see how to assign permissions to execute stored procedures and functions.

Demonstration Steps

1. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to **D:\Demofiles\Mod03**, click **AuthorizingUsersToExecuteCode.sql**, and then click **Open**.
3. Execute the code under the heading for **Step 1** to change database context.
4. Execute the code under the heading for **Step 2** to change execution context.
5. Execute the code under the heading for **Step 3** to try to execute the **uspGetManagerEmployees** stored procedure. Note that permission is denied.
6. Execute the code under the heading for **Step 4** to revert the execution context.
7. Execute the code under the heading for **Step 5** to grant EXECUTE permissions for the stored procedure.
8. Execute the code under the heading for **Step 6** to change execution context.
9. Execute the code under the heading for **Step 7** to try to execute the **uspGetManagerEmployees** stored procedure again. Note that this time the code executes.
10. Execute the code under the heading for **Step 8** to try to execute the **ufnGetStock** function. Note that permission is denied.
11. Execute the code under the heading for **Step 9** to revert the execution context.
12. Execute the code under the heading for **Step 10** to grant EXECUTE permissions on the function.
13. Execute the code under the heading for **Step 11** to change the execution context and test the new permission. Note that the function now works as expected.
14. On the **File** menu, click **Close**.
15. Leave SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
What permission enables a user to change the definition of a stored procedure?	
Select the correct answer.	
	CHANGE
	CHANGE DEFINITION
	ALTER
	ALTER DEFINITION

Lesson 3
Configuring Permissions at the Schema Level

Schemas are used as containers for objects such as tables, views, and stored procedures. They can be particularly helpful in providing a level of organization and structure when large numbers of objects exist in a database. You can also assign security permissions at the schema level, rather than individually on the objects contained in the schemas. Doing this can greatly simplify the design of system security requirements.

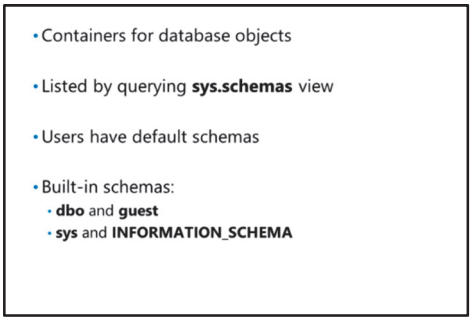
Lesson Objectives

After completing this lesson, you will be able to:

- Describe user-schema separation.
- Describe the role of object name resolution.
- Grant permissions at schema level.

Overview of User-Schema Separation

You can use schemas to contain objects—such as tables, functions, stored procedures, views, and so on—and to provide a security boundary for the assignment of permissions. You can create a schema by using the CREATE SCHEMA statement—the name of the schema forms part of the multipart naming convention for objects. The full name of any object is built up as Server.Database.Schema.Object. You can return a list of all the schemas in a database by querying the **sys.schemas** view.



Schemas can simplify the assignment of permissions. For example, if you grant the EXECUTE permission on a schema to User1, they can then execute all stored procedures in the schema. In schemas that contain a large number of objects, this can expedite the permission granting process.

Note: If you are upgrading applications from SQL Server 2000 or earlier versions, note that schemas were not used then, and that the naming convention consisted of Server.Database.Owner.Object. When upgrading databases, SQL Server will automatically create a schema using the same name as the object owner.

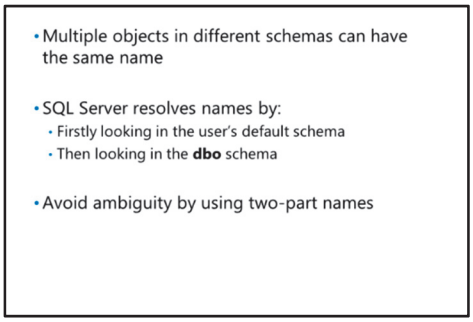
You can assign a default schema to users; this is used when a user refers to an object without specifying a schema name.

There are a few built-in schemas in SQL Server. The **dbo** and **guest** users have associated schemas attached to their own names. The **sys** and **INFORMATION_SCHEMA** schemas are reserved for system objects that you cannot drop or create objects in.

Object Name Resolution

When you refer to object names in code, SQL Server must determine which underlying objects you are referring to.

If a database contains more than one table named Product, each in a different schema, the following code becomes ambiguous:



Referring to Object Names

```
SELECT ProductID, Name FROM Product
```

SQL Server then has to determine which Product table to use. Most users have a default schema that is inferred when a schema is not specified. If a user does not have a default schema, SQL Server assumes the **dbo** schema.

In SQL Server 2012 and later versions, you can assign a default schema to a Windows Group to accommodate users created from certificates.

Note: Creating users from certificates is an advanced topic that is beyond the scope of this course.

When locating an object, SQL Server will first check the user's default schema. If the object is not found, SQL Server will then check the **dbo** schema. Therefore, it is important to include schema names when referring to an object, as shown in the following example:

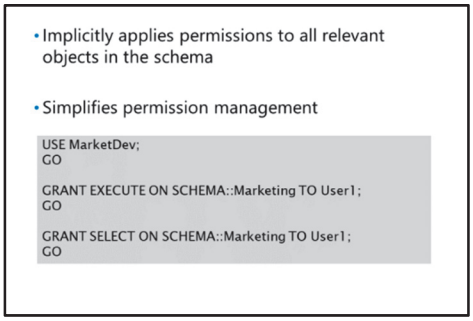
Locating Objects

```
SELECT ProductID, Name FROM Production.Product;
```

Apart from rare situations, using multipart names leads to more reliable code that does not depend on default schema settings.

Granting Permissions at Schema Level

Instead of assigning individual permissions on tables, views, and stored procedures, you can grant permissions at the schema levels that apply to all the relevant objects in the schema. This method of assigning permissions can make them much easier to manage than when you assign permissions to individual objects.



In the following example, User1 is granted EXECUTE permissions on the Marketing schema, meaning that they can execute all stored procedures and scalar functions in the schema. They are then granted SELECT permissions on the same schema, so that they can select from all tables, views, and table-valued functions in the schema:

Granting Permissions

```
USE MarketDev;
GO

GRANT EXECUTE ON SCHEMA::Marketing TO User1;
GO

GRANT SELECT ON SCHEMA::Marketing TO User1;
GO
```

Demonstration: Configuring Permissions at the Schema Level

In this demonstration, you will see how to assign and revoke permissions at schema level.

Demonstration Steps

1. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to **D:\Demofiles\Mod03**, click **ConfiguringPermissionsAtSchemaLevel.sql**, and then click **Open**.
3. Execute the code under the heading for **Step 1** to change database context.
4. Execute the code under the heading for **Step 2** to revoke permission on the **uspGetManagerEmployees** stored procedure.
5. Execute the code under the heading for **Step 3** to confirm that permission was revoked.
6. Execute the code under the heading for **Step 4** to grant EXECUTE permissions on the **dbo** schema.
7. Execute the code under the heading for **Step 5** to try to confirm that permission is now granted on the schema and the stored procedure in it.
8. Execute the code under the heading for **Step 6** to deny permission to execute the stored procedure.
9. Execute the code under the heading for **Step 7** to confirm that permission is denied.
10. Execute the code under the heading for **Step 8** to change database context.
11. Execute the code under the heading for **Step 9** to create a new function.
12. Execute the code under the heading for **Step 10** to explore which permissions imply the ability to select from a schema.
13. Execute the code under the heading for **Step 11** to explore which permissions imply the ability to view the definition of an object.
14. Execute the code under the heading for **Step 12** to explore which permissions imply the ability to select from an object.
15. Execute the code under the heading for **Step 13** to drop the user and the login.
16. On the **File** menu, click **Close**.
17. Close SQL Server Management Studio.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? If a user does not have a default schema, SQL Server assumes the **guest** schema.	

Lab: Authorizing Users to Access Resources

Scenario

The MIA-SQL SQL Server instance will be used to host application databases, including the **InternetSales** database that contains the following schemas and database objects:

- **Sales** schema:
  - **SalesOrderHeader** table
  - **SalesOrderDetail** table
- **Products** schema:
  - **Product** table
  - **ProductSubcategory** table
  - **ProductCategory** table
  - **vProductCatalog** view
  - **ChangeProductPrice** stored procedure
- **Customers** schema:
  - **Customer** table

The **InternetSales** database must be accessible by the following users:

- IT support personnel.
- Sales employees in North America, Europe, and Asia.
- Sales managers.
- An e-commerce web application that runs as the **adventureworks\WebApplicationSvc** service account.

The **adventureworks.msft** domain includes the following global groups:

- **adventureworks\IT_Support**: contains all IT support personnel.
- **adventureworks\Sales_Asia**: contains all sales employees in Asia.
- **adventureworks\Sales_Europe**: contains all sales employees in Europe.
- **adventureworks\Sales_NorthAmerica**: contains all sales employees in North America.
- **adventureworks\Sales_Managers**: contains all sales managers.

The domain administrator has created the following domain local groups, with the members shown:

- **ADVENTUREWORKS\Database_Managers**:
  - ADVENTUREWORKS\IT_Support
- **ADVENTUREWORKS\InternetSales_Users**:
  - ADVENTUREWORKS\Sales_Asia
  - ADVENTUREWORKS\Sales_Europe
  - ADVENTUREWORKS\Sales_NorthAmerica
- **ADVENTUREWORKS\InternetSales_Managers**:
  - ADVENTUREWORKS\Sales_Managers

A SQL Server administrator has created the following server logins and database users in the **InternetSales** database, mapped to their relevant Windows accounts:

- **Database_Managers**
- **InternetSales_Users**
- **InternetSales_Managers**
- **WebApplicationSvc**

The security requirements for the database are:

- The e-commerce application must be able to read data from the **Products.vProductCatalog** view.
- The e-commerce application must be able to insert rows into the **Sales.SalesOrderHeader** and **Sales.SalesOrderDetail** tables.
- All sales employees and managers must be able to read all data in the **Customers** table.
- Sales managers must be able to execute the stored **ChangeProductPrice** procedure.
- Sales managers must be able to insert and update any data in the **Sales** schema.
- All sales employees and managers must be able to read all data in the **Sales** schema.

Objectives

After completing this lab, you will be able to:

- Grant, deny, and revoke permissions at object level.
- Grant EXECUTE permissions on code.
- Grant permissions at schema level.

Estimated Time: 45 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Granting, Denying, and Revoking Permissions on Objects

Scenario

You have been supplied with a list of security requirements. Some of these need to be met using permissions assigned at the object level. In this exercise, you will assign the required object-level permissions.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Grant Permissions on Objects

3. Deny Permissions on Objects

4. Revoke Permissions on Objects

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. Run **Setup.cmd** in the **D:\Labfiles\Lab03\Starter** folder as Administrator.

▶ Task 2: Grant Permissions on Objects

1. Review the supplied security requirements in the scenario for this lab.
2. Determine the permissions that should be assigned at the object level.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of SQL Server.
4. In a new query window, write and execute a script to grant the permissions for the e-commerce application in the **InternetSales** database.
5. In the query window, write and execute a script to grant the permissions for the **Customer** table.
6. Open a command prompt and enter the following command (which opens the sqlcmd utility as ADVENTUREWORKS\AnthonyFrizzell):

```
runas /user:adventureworks\anthonyfrizzell /noprofile sqlcmd
```

7. When you are prompted for a password, enter **Pa$$w0rd**.
8. In the SQLCMD window, enter the following commands to verify your identity:

```
SELECT suser_name();
GO
```

9. In the SQLCMD window, type and execute Transact-SQL statements to verify that Anthony can select data from the **Customer** table in the **InternetSales** database.

▶ Task 3: Deny Permissions on Objects

1. You realize that the **Database_Managers** do not need to access the customer information, so decide to deny them access.
2. In SQL Server Management Studio, write and execute a statement to deny the **Database_Managers** group SELECT permissions on the **Customer** table.
3. In the SQLCMD window, type and execute Transact-SQL statements to verify that Anthony cannot select data from the **Customer** table in the **InternetSales** database.

▶ Task 4: Revoke Permissions on Objects

1. You realize that, although the **Database_Managers** users do not need to access the customer information, Anthony is a member of another group and therefore does need access to the table. You decide to revoke the deny permission that you have implemented, leaving Anthony to inherit permissions from his other group membership.
2. In SQL Server Management Studio, write and execute a statement to revoke SELECT permissions on the **Customer** table from the **Database_Managers** group.
3. In the SQLCMD window, type and execute Transact-SQL statements to verify that Anthony can access the **Customer** table through his membership of the **Sales_Managers** global group, and hence the **InternetSales_Managers** local group and SQL Server login.
4. Close the SQLCMD window.
5. In SQL Server Management Studio, close the query window without saving any changes.
6. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have assigned the required object-level permissions.

Exercise 2: Granting EXECUTE Permissions on Code

Scenario

You have been supplied with a list of security requirements. Some of these need to be met using EXECUTE permissions on code. In this exercise, you will assign the required permissions.

The main tasks for this exercise are as follows:

1. Grant EXECUTE Permission

2. Test the Permission

▶ Task 1: Grant EXECUTE Permission

1. Review the supplied security requirements in the scenario for this lab.
2. Determine the permissions that should be assigned on code.
3. In a new query window, write and execute a script to grant permission for the sales managers to run the **ChangeProductPrice** stored procedure.

▶ Task 2: Test the Permission

1. In the command prompt window, enter the following command (which opens the sqlcmd utility as ADVENTUREWORKS\deannaball):

```
runas /user:adventureworks\deannaball /noprofile sqlcmd
```

2. When you are prompted for a password, enter **Pa$$w0rd**.
3. In the SQLCMD window, type and execute Transact-SQL statements to verify that Deanna can run the stored procedure.
4. Close the SQLCMD window.
5. In SQL Server Management Studio, write and execute a Transact-SQL statement to verify that the stored procedure updated the price.
6. In SQL Server Management Studio, close the query window without saving any changes.
7. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have assigned the required EXECUTE permissions on stored procedures.

Exercise 3: Granting Permissions at the Schema Level

Scenario

You have been supplied with a list of security requirements. Some of these need to be met using schema-level permissions. In this exercise, you will assign the required permissions.

The main tasks for this exercise are as follows:

1. Grant Permission on a Schema

2. Test the Permission

▶ Task 1: Grant Permission on a Schema

1. Review the supplied security requirements in the scenario for this lab.
2. Determine the permissions that should be assigned at the schema level.
3. In a new query window, write and execute a script to grant permission for the sales managers to insert and update data in the **Sales** schema, and for the sales employees and managers to read data in the **Sales** schema.

▶ Task 2: Test the Permission

1. In the command prompt window, enter the following command (which opens the sqlcmd utility as ADVENTUREWORKS\anthonyfrizzell):

```
runas /user:adventureworks\anthonyfrizzell /noprofile sqlcmd
```

2. When you are prompted for a password, enter **Pa$$w0rd**.
3. In the SQLCMD window, type and execute Transact-SQL statements to verify that Anthony can access, insert, and update sales data.
4. Close the SQLCMD window.
5. Close the command prompt window.
6. In SQL Server Management Studio, close the query window without saving any changes.
7. Close SQL Server Management Studio without saving any changes.

Results: After completing this exercise, you will have assigned the required schema-level permissions.

Question: Your organization needs to track data access by individual Windows users. Does this mean you cannot base logins on Windows groups?

Module Review and Takeaways

In this module, you have seen how to manage objects, code, and schema permissions in a database.

 **Best Practice:** Assigning permissions at schema level can simplify your security architecture.

Review Question(s)

Question: Regarding permissions, how does SQL Server differ from ANSI SQL?

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

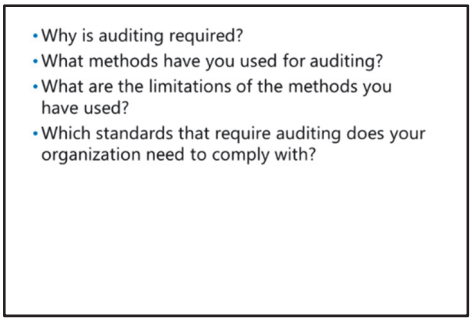
At the end of this lesson, you will be able to:

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

Discussion: Auditing Data Access

During this discussion, you will consider the following questions:

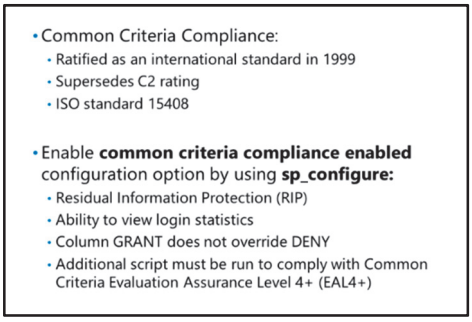
- Why is auditing required?
- What methods have you used for auditing?
- What are the limitations of the methods you have used?
- Which standards that require auditing does your organization need to comply with?



Enabling Common Criteria Compliance

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:

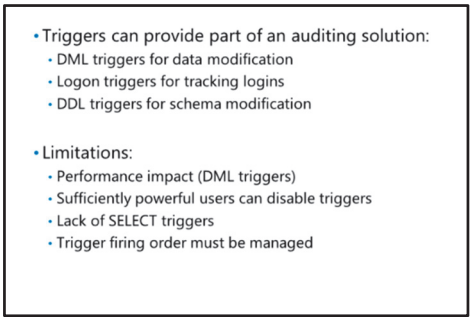
common criteria compliance enabled Server Configuration Option

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:

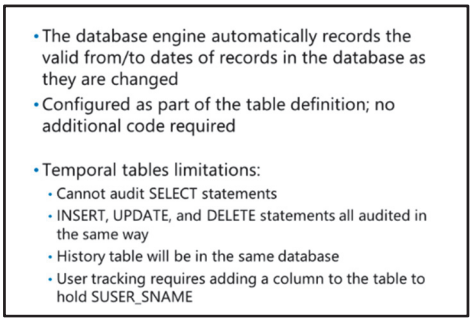
- System performance can be significantly impacted by DML triggers running alongside the usual load on the server.
- Users with appropriate permissions can disable triggers. This can be a significant issue for auditing requirements.
- DML triggers cannot be used to audit data access through SELECT statements.
- 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



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.

Categorize Activity

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

Items	
1	Triggers
2	SQL Server Profiler
3	Temporal Tables

Category 1		Category 2
Records Changes to Data or Database Objects		Records DML Statements or DDL Statements

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.

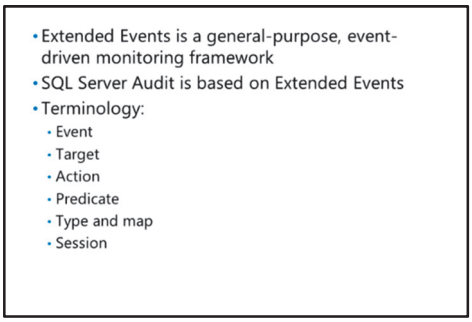
Lesson Objectives

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.

- Server-level audit
  - All editions of SQL Server
- Database-level audit
  - Enterprise, Developer, and Evaluation editions
- Terminology:
  - Server Audit
  - Server Audit Specification
  - Database Audit Specification
  - Actions
  - Action Groups
  - Target

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.

Object	Description
Actions	Specific actions that can raise events and be added to the audit. For example, SELECT operations on a table.
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.

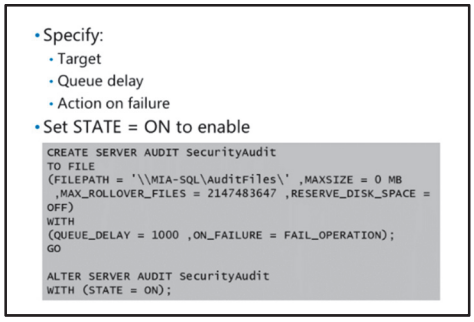
For more information about SQL Server Audit, see *SQL Server Audit (Database Engine)* in Microsoft Docs:

SQL Server Audit (Database Engine)

http://aka.ms/ucupsq

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.

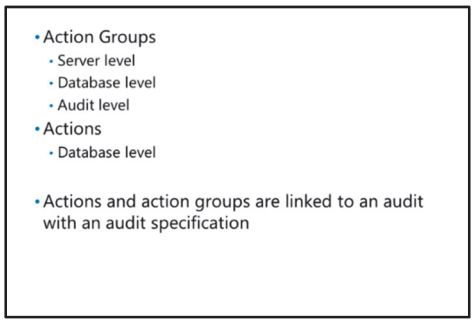
For more information on the CREATE SERVER AUDIT command, see *CREATE SERVER AUDIT (Transact-SQL)* in Microsoft Docs:

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.

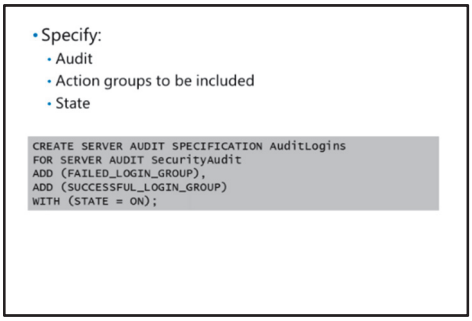
For a full list of available server actions and action groups, see *SQL Server Audit Action Groups and Actions* in Microsoft Docs:

SQL Server Audit Action Groups and Actions

http://aka.ms/bak8rw

Creating Server Audit Specifications

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);
```

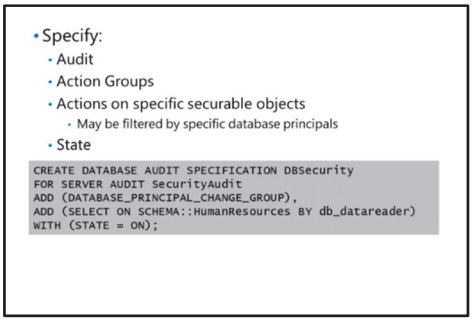
For more information on the CREATE SERVER AUDIT SPECIFICATION command, see *CREATE SERVER AUDIT SPECIFICATION (Transact-SQL)* in Microsoft Docs:

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);
```

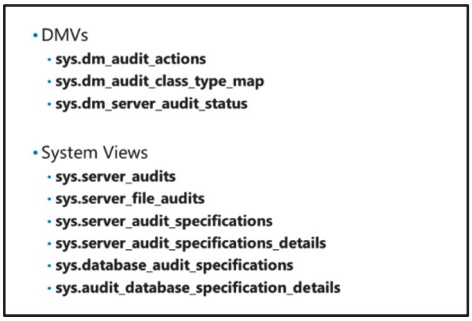
For more information on the CREATE DATABASE AUDIT SPECIFICATION command, see CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL) in Microsoft Docs:

CREATE DATABASE AUDIT SPECIFICATION (Transact-SQL)

http://aka.ms/x6x4qu

Audit-Related Dynamic Management Views and System Views

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:

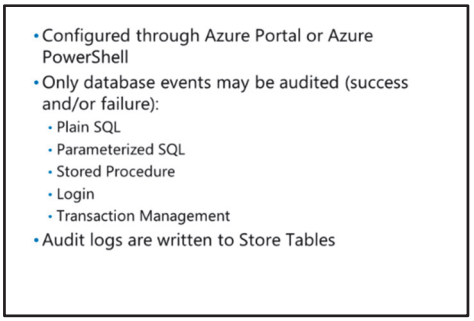
- **sys.server_audits**. Returns a list of all the audits defined on an instance of SQL Server.
- **sys.server_file_audits**. Returns a list of all audits that write data to a file target.
- **sys.server_audit_specifications**. Returns a list of high level information about server audit specifications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

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.

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

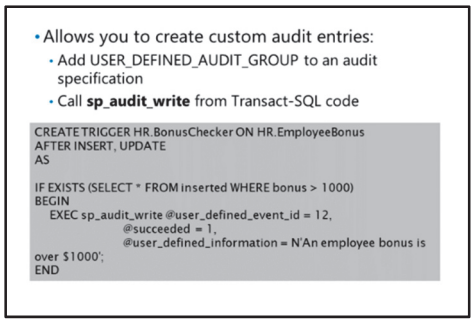
1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and **20764C-MIA-SQL** virtual machines are running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod04** folder as Administrator.
3. In the **User 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.

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

Question	
Which of the following is not a component of the SQL Server Audit architecture?	
Select the correct answer.	
	Target
	Target Group
	Server Audit
	Database Audit Specification
	Server Audit Specification

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.

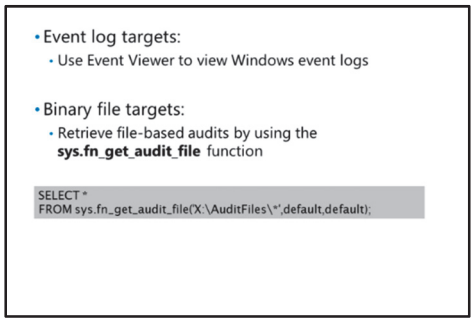
Lesson Objectives

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);
```

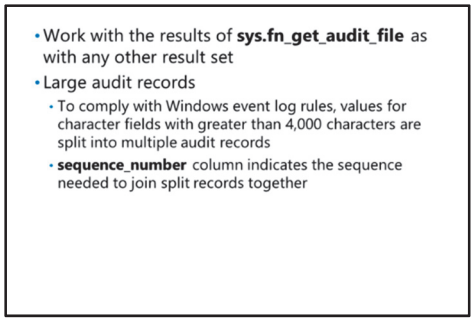
For more information on **sys.fn_get_audit_file**, see *sys.fn_get_audit_file (Transact-SQL)* in Microsoft Docs:

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

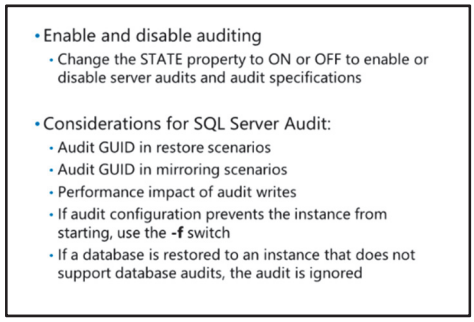
Large Audit Records

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

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

Question	
Which of the following is not a target for SQL Server Audit?	
Select the correct answer.	
	File
	Windows Application Log
	Windows Security Log
	Ring Buffer

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.

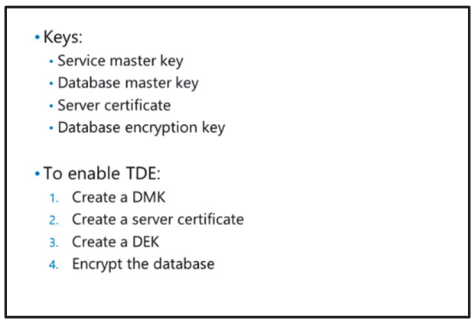
Lesson Objectives

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

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. Detach the source database
2. Copy/move database files
3. Create new SMK in the **master** database of the target server
4. Generate a new server certificate from a backup of the server certificate on the source server, and its associated private key
5. Attach the 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.

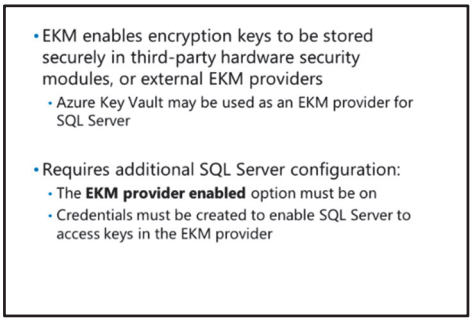
4. Use a CREATE CERTIFICATE Transact-SQL statement to generate a server certificate on the destination server from the backup of the original server certificate and its private key.
5. Attach the database on the destination server.

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)

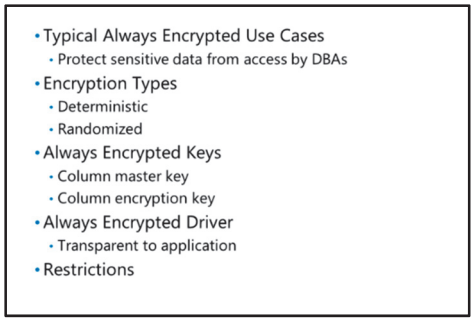
http://aka.ms/dmhvxl

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.

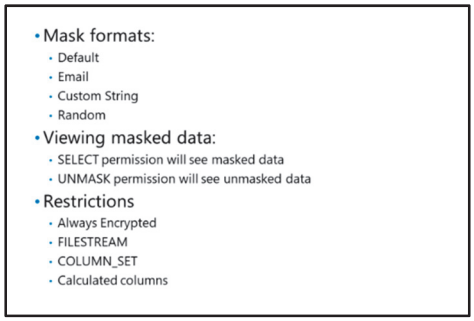
For more information on Always Encrypted, including a full list of restrictions, see *Always Encrypted (Database Engine)* in Microsoft Docs:

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

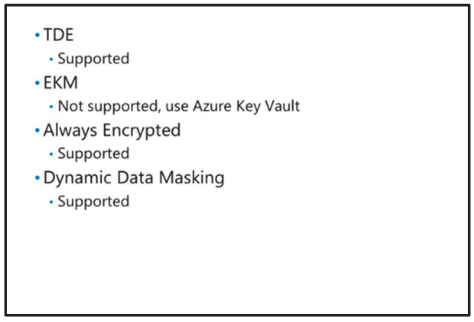
```
bank_account varchar(50) MASKED WITH (FUNCTION = 'default()') NULL
```

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



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.

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.

Items	
1	Data encrypted at rest
2	Data encrypted at rest and in transit
3	Data values obfuscated
4	Encryption and decryption carried out by SQL Server
5	Encryption and decryption take place at client application
6	Data stored in plain text
7	Acts at database level
8	Acts at column level

Category 1		Category 2		Category 3
TDE		Always Encrypted		Dynamic Data Masking

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.

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

▶ Task 1: Prepare the Lab Environment

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:\Labfiles\Lab04\Starter** folder as Administrator.

▶ Task 2: Create a Server Audit

1. Using the SQL Server Management Studio (SSMS) GUI, create a server audit with the following properties:
   - **Name**: activity_audit
   - **Queue delay**: 1000 ms
   - **On failure**: continue
   - **Target**: file
   - **Target file path**: D:\Labfiles\Lab04\Starter\Audit
2. Enable the server audit you have created.

▶ Task 3: Create a Server Audit Specification

1. Using the SSMS GUI, create a server audit specification with the following properties:
   - **Name**: audit_logins
   - **Audit**: activity_audit
   - **Action groups**: SUCCESSFUL_LOGIN_GROUP
2. Enable the server audit specification you have created.

▶ Task 4: Create a Database Audit Specification

1. Use the SSMS GUI to create a database audit specification on the **salesapp1** database with the following properties:
   - **Name**: employees_change_audit
   - **Audit**: activity_audit
   - **Actions**:
     a. INSERT on **HR.Employees** by **public**
     b. UPDATE on **HR.Employees** by **public**
2. Enable the database audit specification you have created.

▶ Task 5: Generate Audited Activity

1. In SSMS, open the project file **D:\Labfiles\Lab04\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - audit activity.sql**.
2. Execute the code under the heading for **Task 1** to generate some activity that will appear in the audit log.

▶ Task 6: Review Audit Data

- Under the heading for **Task 2**, write a query using the **sys.fn_get_audit_file** system function to return all the audit data from files in **D:\Labfiles\Lab04\Starter\Audit**. Filter the data so that only activity related to your current session is displayed.

▶ Task 7: Disable the Audit

- Using the SSMS GUI, disable the **activity_audit** server audit.

Results: 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.

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

▶ Task 1: Encrypt a Column

1. Using the SSMS GUI, configure the **phone** row of the **salesapp1.Sales.Customers** table to be encrypted using Always Encrypted.
2. Run a SELECT statement against the **salesapp1.Sales.Customers** table to view the encrypted data.

▶ Task 2: View Always Encrypted Data from an Application

1. Run the PowerShell script **query_encrypted_columns.ps1** in **D:\Labfiles\Lab04\Starter**.
2. Review the output of the script. The script demonstrates how a change in the connection string to enable the **Column Encryption Setting** property allows it to decrypt the Always Encrypted column. This is possible because the script has access to the column master key in the local Windows key store.

Results: 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

▶ Task 1: Create a Service Master Key

1. In SSMS Solution Explorer, open the query file **Lab Exercise 03 - TDE.sql**.
2. Execute the code under the heading for **Task 1** to create a service master key.

▶ Task 2: Back Up the Service Master Key

- Under the heading for **Task 2**, edit the query to back up the service master key to **D:\Labfiles\Lab04\Starter\Audit\smk.bak**. Encrypt the backup with the password **iD2Z1i85saFyAiK7auzn$**.

▶ Task 3: Create and Back Up a Server Certificate

1. Execute the code under the heading for **Task 3** to create a certificate called **TDE_cert**.
2. Under the heading for **Task 4**, edit the query to back up the **TDE_cert** server certificate to **D:\Labfiles\Lab04\Starter\Audit\TDE_cert.bak**, and to back up the **TDE_cert** private key to **D:\Labfiles\Lab04\Starter\Audit\TDE_cert_pk.bak**, using the password ***R8vkULA5aKhp3ekGg1o3**.

▶ Task 4: Create a Database Encryption Key and Encrypt the salesapp1 Database

1. Under the heading for **Task 5**, edit the query to create a database encryption key using the **AES_256** algorithm, encrypted by the server certificate **TDE_cert**.
2. Execute the code under the heading for **Task 6** to encrypt the **salesapp1** database.
3. Execute the code under the heading for **Task 7** to examine the **sys.dm_database_encryption_keys** DMV. Notice that both **salesapp1** and **tempdb** have been encrypted—**tempdb** is encrypted if any database on the instance is encrypted.

▶ Task 5: Move the salesapp1 Database

1. Detach the **salesapp1** database from the **MIA-SQL** instance.
2. Attempt to attach the **salesapp1** database to the **MIA-SQL\SQL2** instance.
   - The primary database file is named **salesapp1.mdf**, in the **D:\Labfiles\Lab04\Starter\Setupfiles** folder.
   - Attaching the database should fail because the certificate with which the database encryption key is protected does not exist on the **MIA-SQL\SQL2** instance.
3. Create a server master key for the **MIA-SQL\SQL2** instance.
4. Create a certificate named **TDE_cert** in the master database on the **MIA-SQL\SQL2** instance from the backup certificate and private key files you created previously.
5. Attach the **salesapp1** database to the **MIA-SQL\SQL2** instance. Verify that you can access the data it contains.

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

Encrypt a database using TDE.

Move an encrypted database to another SQL Server instance.

Check Your Knowledge

Question	
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)?	
Select the correct answer.	
	Deterministic encryption
	Randomized encryption
	Neither of the above

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)

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

Module 5

Recovery Models and Backup Strategies

Contents:

Module Overview

One of the most important aspects of a database administrator's role is ensuring that organizational data is reliably backed up so that, if a failure occurs, you can recover the data. Even though the computing industry has known about the need for reliable backup strategies for decades—and discussed this at great length—unfortunate stories regarding data loss are still commonplace. A further problem is that, even when the strategies in place work as they were designed, the outcomes still regularly fail to meet an organization's operational requirements.

In this module, you will consider how to create a strategy that is aligned with organizational needs, based on the available backup models, and the role of the transaction logs in maintaining database consistency.

Objectives

After completing this module, you will be able to:

- Describe various backup strategies.
- Describe how database transaction logs function.
- Plan SQL Server backup strategies.

Lesson 1
Understanding Backup Strategies

SQL Server supports three database recovery models. All models preserve data in the event of a disaster, but there are important differences that you need to consider when selecting a model for your database.

Choosing the appropriate recovery model is an important part of any backup strategy. The recovery model that you select for your database will determine many factors, including:

- Maintenance processing overhead.
- Exposure to potential loss.
- The available backup types.

When choosing a recovery model for your database, you will need to consider the size of the database, the potential maintenance overhead, and the level of acceptable risk regarding potential data loss.

Lesson Objectives

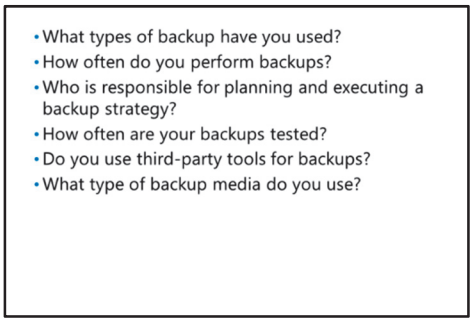
After completing this lesson, you will be able to:

- Determine appropriate backup strategies.
- Choose appropriate backup media.
- Establish backup retention policies.
- Understand SQL Server Backup with Azure Data Blob storage.

Discussion: Previous Experience with Backup Strategies

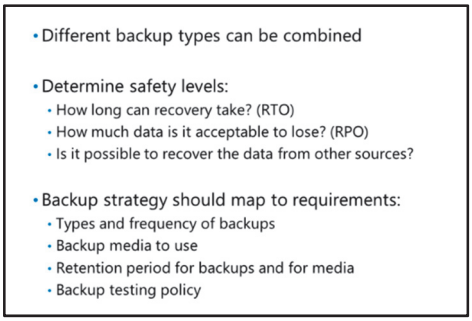
In this discussion, members of your class will share their experiences with backup strategies. In particular, consider (but do not limit yourself to) the following questions:

- What types of backup have you used?
- How often do you perform backups?
- Who is responsible for planning and executing a backup strategy?
- How often are your backups tested?
- Do you use third-party tools for backups?
- What type of backup media do you use?



Determining an Appropriate Backup Strategy

SQL Server provides a variety of backup types. Usually, there is no single backup type available to satisfy an organization's requirements when forming a backup strategy. More commonly, a combination of backup types is required to achieve an appropriate outcome. Later in this module, and in the next module, you will learn about the specific types of backups that SQL Server provides.



Key Criteria

When designing a backup strategy, there is always a tradeoff between the level of safety that is guaranteed and the cost of the solution. If you ask any business about how much data they can afford to lose, you will almost certainly be told that they cannot afford to lose any data, in any circumstances. Yet, while no data loss is an admirable goal, it is not affordable or realistic. For this reason, there are two objectives that need to be established when discussing a backup strategy: a recovery time objective (RTO) and a recovery point objective (RPO). Part of the strategy might also involve the retrieval of data from other locations where copies of the data are stored.

Recovery Time Objective

There is little point in having perfectly recoverable data if the time taken to recover that data is too long. A backup strategy needs to have an RTO. For example, consider the backup requirements of a major online bank. How long could a bank tolerate a situation where it was unable to access any of the data in its systems? Now imagine that the bank was making full copies of all its data, yet a full restore of the data would take two weeks to complete. What impact would a two-week outage have on the bank? The more important question is how long would an interruption to data access need to be, before the bank ceased to be viable?

The key message with RTO is that a plan that involves quick recovery, with a small data loss, might be more palatable to an organization than a plan that reduces data loss, but takes much longer to implement. Another key issue is that the time taken to restore data might also involve finding the correct backup media; finding a person with the authority to perform the restore; finding documentation related to the restore, and so on.

Recovery Point Objective

After a system has been recovered, hopefully in a timely manner, the next important question relates to how much data has been lost. This is represented by the RPO. For example, while a small business might conclude that restoring a backup from the previous night, with the associated loss of up to a day's work is an acceptable risk tradeoff, a large business might see the situation very differently. It is common for large corporations to plan for zero committed data loss. This means that work that was committed to the database must be recovered, but that it might be acceptable to lose work that was in process at the time of the failure.

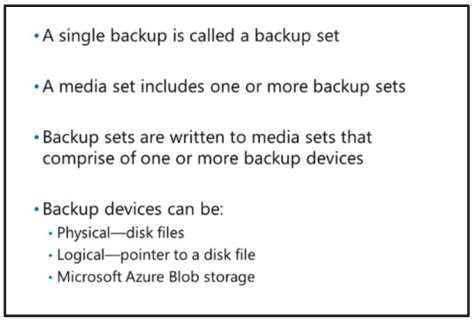
Mapping to Business Strategy

The most important aspect of creating a backup strategy is that it must be designed in response to the business requirements and strategy. The backup strategy also needs to be communicated to the appropriate stakeholders within the organization. It is important to make sure that the expectations of the business users are managed, in line with the agreed strategy.

Organizations often deploy large numbers of databases. The RPO and RTO for each database might be different. This means that database administrators will often need to work with different backup strategies for different databases that they are managing. Most large organizations have a method of categorizing the databases and applications, in terms of importance to the core functions of the organization. The business requirements will determine all aspects of the backup strategy, including how frequently backups need to occur; how much data is to be backed up each time; the type of media that the backups will be held on; and the retention and archival plans for the media.

Choosing Appropriate Backup Media

A backup set contains the backup from a single, successful backup operation performed by SQL Server. One or more backup sets are written to a media set, which is represented by a file at the operating system or device level. It is important to realize that, because a single file (or media set) can contain more than one backup, when it is time to restore a backup, you need to ensure that you are restoring the intended backup from within the file.



Physical Backup Devices

SQL Server supports the creation of backups to disk files. UNC file paths are supported for disk locations, so that backups can be written to network file shares. Earlier versions of SQL Server supported writing backups directly to tape, but that option is now deprecated and should not be used for new development. It is generally considered better practice to write a backup to disk first, and to copy the disk backup to tape later, if required.

The backups that SQL Server creates are encoded in a format known as Microsoft Tape Format (MTF)—a common format that is used by other Microsoft products in addition to SQL Server. This means that SQL Server backups can be combined with other backups, such as operating system backups, on the same media sets.

Note: Compressed SQL Server backups cannot share media with other types of backup.

Logical Backup Devices

You can specify an operating system filename that a backup should be written to, directly in the BACKUP command, or in the GUI within SSMS. However, SQL Server also allows a degree of indirection by the creation of logical backup devices. A logical backup device is an optional user-defined name that refers to a specific output location.

By using logical backup devices, an application can be designed to always send backups to the logical backup device, instead of to a specific physical location.

For example, an HR application could be designed to create backups on a logical backup device named HRBackupDevice. A database administrator could later determine where the backups should be physically sent. The administrator could then decide the name of a file that should hold the backups from the HR application. No changes would need to be made to the HR application to accommodate the change in backup file location, as the application would always back up to the same logical backup device.

Backup Mirroring and Striping

A single backup can target more than one backup device. Up to 64 devices are supported for a single media set. If more than one backup device is used, the backups can be mirrored or striped.

With a mirrored backup (only available in Enterprise edition), the same backup data is written to each backup device concurrently. This option provides for redundancy of the physical backup device. Only one of the devices needs to be present during a restore process.

Note: While mirrored backups help provide fault tolerance regarding media failure after the backup completes, mirrored backups are actually a fault-intolerant option during the backup process. If SQL Server cannot write to one of the mirrored devices, the entire backup fails.

Striping of backups causes a single backup to be written across a set of backup devices. Each backup device receives only part of the backup. All backup devices need to be present when a restore of the data is required.

Required Privileges

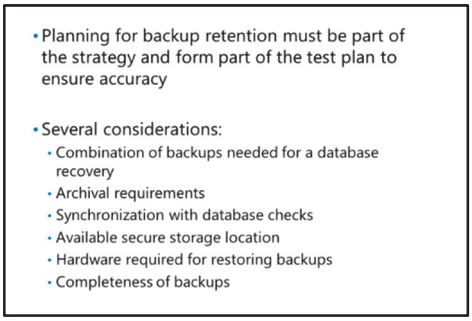
To perform a backup, you must be a member of the sysadmin fixed server role or the **db_owner** or **db_backupoperator** fixed database roles.

Determining a Retention Policy for Backups

A backup strategy must include plans for retention of backups, and for the locations where the media should be retained.

Too often, organizations regularly perform backup operations but, when the time comes to restore the backups, a restore is not possible. Most of these problems would be alleviated by a good retention and testing plan.

The following list shows examples of the most common problems that are seen and the appropriate avoidance measure:



Insufficient Copies of Backups

Your organization will depend on the quality of backups if they need to be restored. The more copies of backups that you have, and the more pieces of media that are holding all the required data, the better the chance there is of being able to recover.

The act of creating a backup over your most recent backup is generally regarded as the worst offence. If the system fails during the backup, you will often lose both your data and the backup. Consider the example of a database administrator who asked for help on a Microsoft SQL Server forum. The DBA had inadvertently performed a restore operation instead of a backup operation—and the last backup was performed a year ago. Unfortunately, there was little that anyone could do to help recover the situation.

Avoidance strategy: make multiple copies of backups.

Insufficient Data on the Backups

Company A performed regular backups, but recovery had not been tested. The first time that a real recovery was attempted, it was discovered that not all files that needed to be backed up were in fact backed up.
Avoidance strategy: regular reconstruction of data from backup recovery testing.

Unreadable Backups

Company B performed regular backups but did not test them. When recovery was attempted, none of the backups were readable. This situation is often caused by hardware failures, in addition to the inappropriate storage of media.
Avoidance strategy: regular backup recovery testing.

Unavailable Hardware

Company C purchased a special tape drive to perform their backups. When they decided to restore the backups, that special tape drive no longer worked and no other device within the organization could read the backups—even if the backups were valid.
Avoidance strategy: regular backup recovery testing.

Old Hardware

Company D performed regular backups and retained them for an appropriate period. When the company wanted to restore the backups, they no longer possessed equipment that was capable of restoring them.
Avoidance strategy: regular backup recovery testing, combined with recovery and backup onto current devices.

Misaligned Hardware

Company E performed regular backups and even tested that they could perform restore operations from the backups. However, because they tested the restores on the same device that performed the backups, they did not realize that the device was misaligned—and that this was the only device that could read those backups. When a restore was needed, the device that the backups were performed on failed.
Avoidance strategy: regular backup recovery testing on a separate system and a separate physical device.

General Considerations

When multiple types of backups are being performed, it is important to work out the combination of backups that will be needed when a restore is required.

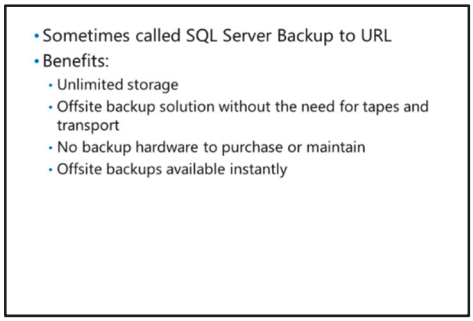
Organizations might need to fulfill legal or compliance requirements regarding the retention of backups. In most cases, full database backups are kept for a longer period of time than other backup types.

Checking the consistency of databases by using DBCC CHECKDB is a crucial part of database maintenance, and is discussed later in this course.

You will need to determine, not only how long backups should be kept, but also where they are kept. Part of the RTO needs to consider how long it takes to obtain the physical backup media, if it needs to be restored.

You also need to make sure that backups are complete. Are all files that are needed to recover the system (including external operating system files) being backed up?

SQL Server Backup with Azure Blob Storage



You can back up an on-premises SQL Server database to Microsoft Azure Blob storage in the same way you would back up to a local storage solution. The many benefits of backing up to Microsoft Azure Blob include:

- Unlimited storage for your backups.
- Offsite backup storage without the need for tapes and transportation.
- No backup hardware to purchase or maintain.
- Offsite backups are available for restore instantly without the need for tape retrieval.

SQL Server backup to Azure Blob storage is sometimes referred to as SQL Server Backup to URL. Implementing SQL Server Backup to Azure Blob storage is as simple as setting up an Azure storage account, configuring the container, and specifying a URL as the backup destination when you back up your database.

Demonstration: Back Up an On-premises Database to Microsoft Azure

In this demonstration, you will see how to back up a database to Azure Blob storage.

Demonstration Steps

Install the Azure PowerShell Module

1. On start menu, type **Windows PowerShell**. Then right-click **Windows PowerShell™**, and click **Run ISE as Administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. At the command prompt, type **Install-Module AzureRM -AllowClobber**, and then press Enter.
4. If the **NuGet provider is required to continue** is displayed, type **Y**, and then press **enter**.
5. If the **Untrusted repository** message is displayed, type **A**, and then press **enter**.
6. Wait until the installation completes, and then close the PowerShell window.

Create a Storage Account

1. Run **Setup.cmd** in the **D:\Demofiles\Mod05** folder as Administrator. In the **User Account Control** dialog box, click **Yes**.
2. On the taskbar, click **Internet Explorer**, and go to **portal.azure.com**.
3. Log into your Azure account, in the left blade, click **Storage accounts**.
4. On the **Storage accounts** blade, click **Add**.
5. On the **Create storage account** blade, enter the following details, and then click **Create**:
    a. **Name**: 20764+yourinitials+TodaysDate, for example 20764jhd20160421
    b. **Deployment model**: Resource manager
    c. **Account kind**: Storage (general purpose v1)

   d. **Performance**: Standard
   e. **Replication**: Zone-redundant storage
   f. **Subscription**: Azure pass
   g. **Resource Group**: 20764C
   h. **Location**: Your nearest location
6. The storage account will be created, minimize Internet Explorer.

Create a Storage Container and SAS Token

1. On the taskbar, right-click the **Windows PowerShell** icon, and then click **Windows PowerShell ISE**.
2. On the **File** menu, click **Open**.
3. In the **Open** dialog box, go to **D:\Demofiles\Mod05**, click **ContainerSAS.ps1**, and then click **Open**.
4. Amend the **$accountName** to the Microsoft account that is associated with your Azure pass.
5. Amend the **$storageAccountName** to the name of the Storage Account you created in the previous task.
6. On the toolbar, click **Run Script**.
7. In the **Windows PowerShell ISE** dialog box, click **OK**.
8. In the **Sign in to your account** dialog box, enter your Microsoft Azure account user name and password, and then click **Sign in**.
9. In the **Confirm** dialog box, click **Yes** to delete the account (don't worry—this doesn't actually delete the account).

   Note: Leave the window open—you will need the information displayed in the next task.

Back Up a Database with Azure Blob Storage

1. Start SQL Server Management Studio, and connect to the **MIA-SQL** instance.
2. On the **File** menu, point to **Open**, and then click **File**.
3. In the **Open File** dialog box, go to **D:\Demofiles\Mod05**, click **AzureBackup.sql**, and then click **Open**.
4. Replace the two instances of **https://xxxx.blob.core.windows.net/aw2018** entries on line 2 and line 11 to the name of your **Cloud Blob Container URL** in the PowerShell window.
5. Amend the **sv=enter key here** entry to your **Shared Access Signature** in the PowerShell window
6. In SQL Server Management Studio, highlight the statement under the comment **Create credential**, and then click **Execute**.
7. Highlight the statements underneath the comment **Backup the database**, click **Execute**, and wait for the backup process to complete successfully.
8. In Internet Explorer, on the **All resources** blade, click **Refresh**, and then click the name of your **Storage account (classic)**.
9. On your account blade, under **BLOB SERVICE**, click **Containers**, and then click **aw2016**.
10. Verify that the **logtest.bak** backup file has been created.
11. Close Internet Explorer and close Windows PowerShell.
12. Leave SSMS open for the next demonstration.

Question: What are the advantages of using SQL Server Backup with Azure Blob storage?

Lesson 2
SQL Server Transaction Logs

Before you can plan a backup strategy, you must understand how SQL Server uses the transaction log to maintain data consistency, and how the recovery model of the database affects transaction log operations, in addition to the available backup options.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain how the transaction log functions.
- Describe the transaction log file structure.
- Understand SQL Server recovery models.
- Understand capacity planning for transaction logs.
- Work with checkpoint options.

Overview of SQL Server Transaction Logs

Transaction logs provide a history of actions executed by a database management system to guarantee atomicity and durability of transactions:

1. Data modification is sent by the application
2. Data pages are located in or read into the buffer cache and then modified
3. Modification is recorded in the transaction log on disk
4. Later, checkpoint writes dirty pages to data files

Two of the common requirements for transaction management in database management systems are the atomicity and durability of transactions. Atomicity requires that an entire transaction is committed or that no work at all is committed. This is common in banking transaction, where money moves from one account to another—two updates are required to be contained within a single atomic transaction to complete the transfer. Durability requires that, once a transaction is committed, it will survive system restarts, including those caused by system failures. SQL Server uses the transaction log to ensure both the atomicity and durability of transactions.

Write-Ahead Logging

When SQL Server needs to modify the data in a database page, it first checks if the page is present in the buffer cache. If the page is not present, it is read into the buffer cache. SQL Server then modifies the page in memory, writing redo and undo information to the transaction log. While this write is occurring, the "dirty" page in memory is locked until the write to the transaction log is complete. At regular intervals, a background checkpoint process flushes the dirty pages to the database, writing all the modified data to disk.

This process is known as write-ahead logging (WAL) because all log records are written to the log before the affected dirty pages are written to the data files and the transaction is committed. The WAL protocol ensures that the database can always be set to a consistent state after a failure. This recovery process will be discussed in detail later in this course—its effect is that transactions that were committed before the failure occurred are guaranteed to be applied to the database. Those transactions that were "in flight" at the time of the failure, where work is partially complete, are undone.

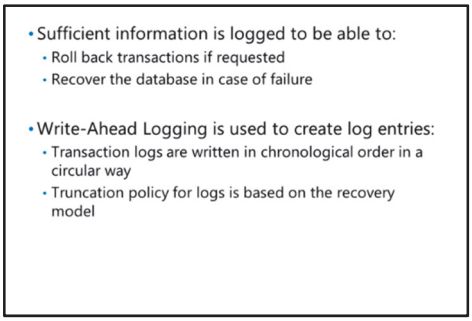
Writing all changes to the log file in advance also makes it possible to roll back transactions if required.

Transaction Rollback

SQL Server can use the information in the transaction log to roll back transactions that have only been partially completed. This ensures that transactions are not left in a partially-completed state. A transaction rollback may occur because of a request from a user or client application (such as the execution of a ROLLBACK TRANSACTION statement) or because a transaction is in a partially-completed state at the time of a system failure.

Transaction Log File Structure

It is essential that there is enough information in the transaction log to process rollback requests from users or applications and to meet the needs of other SQL Server features, such as replication and change data capture.



Transaction Log Structure and Virtual Log Files

SQL Server writes to the transaction log in chronological order, in a circular fashion. Because the file may need to grow, it is internally divided into a set of virtual log files (VLFs) that have no fixed size. When the database engine is creating or extending a log file, it dynamically determines the appropriate size for the VLFs, depending on the size of growth that is occurring. The values in the following table show how it calculates the number of VLFs to create:

Growth Increment	Number of VLFs
Less than or equal to 64 MB	4
Between 64 MB and 1 GB	8
Greater than 1 GB	16

When a log file write reaches the end of the existing log file, SQL Server starts writing again at the beginning, overwriting the log records currently stored. This mechanism works well, providing that the previous log records in that section of the log file have already been written to the database and freed up, or "truncated". If they have not been truncated and the data is required, SQL Server tries to grow the size of the log file. If this is not possible (for example, if it is not configured for automatic growth or the disk is full) SQL Server fails the transaction and returns an error. If it is possible to grow the log file, SQL Server allocates new virtual log files, using the auto growth size increment in the log file configuration.

Note: Instant File Initialization (IFI) cannot be used with transaction log files. This means that transactions can be blocked while log file growth occurs.

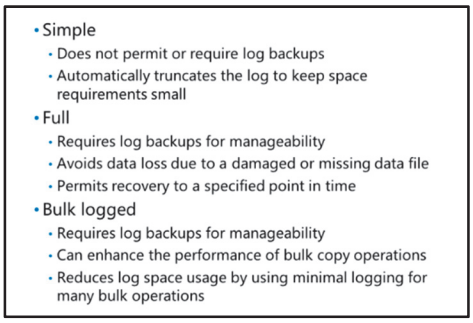
Log File Truncation

Truncation occurs either as part of the backup process for a transaction log or automatically, when using certain database recovery models. The entries in the log file are logically ordered by their Log Sequence Number (LSN), with the starting point of the oldest active transaction being the MinLSN value. When the log file is truncated, only data up to the MinLSN can be truncated. Any entries with an LSN greater than the MinLSN must be retained for use in any potential recovery process. During truncation, only data up to the start of the virtual log file that contains the minimum of the start of the last checkpoint operation, MinLSN, and oldest transaction that is not yet replicated (if using replication), is truncated.

Note: Replication is beyond the scope of this course but it is important to be aware that the configuration and state of replicated data can affect transaction log truncation.

Working with Recovery Models

SQL Server has three database recovery models. All models will preserve data in the event of a disaster but there are important differences that need to be considered by the database administrator when selecting a model for their database.



Choosing the appropriate recovery model is an important part of your recovery strategy. The recovery model that you select for your database will determine many factors including:

- Maintenance processing overhead.
- Exposure to potential loss.
- Which backup types are available.

When choosing a recovery model for the database, you will need to consider the size of the database, the potential maintenance overhead, and the level of acceptable risk with regards to potential data loss.

Simple Recovery Model

The simple recovery model minimizes administrative overhead for the transaction log, because the transaction log is not backed up. The simple recovery model risks significant work-loss exposure if the database is damaged. Data is only recoverable to the point of the most recent backup of the database that has been lost. Therefore, under the simple recovery model, the backup intervals should be short enough to prevent the loss of significant amounts of data. However, the intervals should be long enough to keep the backup overhead from affecting production work. The inclusion of differential backups into a backup strategy, based on the simple recovery model, can help reduce the overhead.

In earlier versions of SQL Server, simple recovery model was referred to as "truncate log on checkpoint". The name was changed to provide a focus on the recovery options, rather than on the process involved in implementing the option. Each time a checkpoint process occurs, SQL Server will automatically truncate the transaction log up to the end of the VLF, before the VLF that contains the MinLSN value. This means that the only role that the transaction log ever plays is the provision of active transaction log data during the recovery of a database.

Full Recovery Model

The full recovery model provides the normal database maintenance model for databases where durability of transactions is necessary. Full recovery model is the default recovery model when SQL Server is installed. The recovery model for new databases is based on the recovery model of the **model** database, which could be changed for sites that wish to use a different default recovery model.

With full recovery model, log backups are required. This recovery model fully logs all transactions and retains the transaction log records until after they are backed up. The full recovery model allows a database to be recovered to the point of failure, assuming that the tail of the log can be backed up after the failure. The full recovery model also supports an option to restore individual data pages or to restore to a specific point in time.

Bulk-logged Recovery Model

This recovery model can reduce the transaction logging requirements for many bulk operations. It is intended solely as an adjunct to the full recovery model. For example, while executing certain large-scale bulk operations, such as bulk import or index creation, a database can be switched temporarily to the bulk-logged recovery model. This temporary switch can increase performance by only logging extent allocations and reducing log space consumption.

Transaction log backups are still required when using bulk-logged recovery model. Like the full recovery model, the bulk-logged recovery model retains transaction log records until after they are backed up. The tradeoffs are bigger log backups and increased work-loss exposure because the bulk-logged recovery model does not support point-in-time recovery.

Note: One potentially surprising outcome is that the log backups can often be larger than the transaction logs. This is because SQL Server retrieves the modified extents from the data files while performing a log backup for minimally-logged data.

Capacity Planning for Transaction Logs

The recovery model that you choose for a database will impact the size of the log file. When using the simple recovery model, SQL Server truncates the log after each checkpoint. In the full and bulk-logged recovery models, the log is truncated after each log backup, to ensure that an unbroken chain of backup log files exists. The truncation of a log file happens after a log backup. There is a common misconception that a full database backup breaks this chain of log file backups—but this is not true.

- Capacity needs are based on several factors:
  - Recovery model used for the database
  - Transaction log backup frequency in full and bulk logged recovery models
  - Number and size of transactions in the database
- Examine log behavior during predeployment testing

Determining Log File Size

It is very difficult to calculate the size requirements for log files. As with planning other aspects of SQL Server, monitoring during realistic testing is the best indicator.

Another common misconception is that the log file of a database in simple recovery model will not grow. This is also not the case. In simple recovery model, the transaction log needs to be large enough to hold all details from the oldest active transaction. Large or long-running transactions can cause the log file to need additional space.

Inability to Truncate a Log

Installed or in-use SQL Server features can also prevent you from truncating log files. For example, database mirroring, transactional replication, and change data capture can all affect the ability for the database engine to truncate log files.

 **Note:** Database mirroring, transactional replication, and change data capture are beyond the scope of this course.

You can use the **log_reuse_wait_desc** column in the **sys.databases** table to identify the reason why you cannot truncate a log.

Identifying Truncation Issues

```
SELECT name, log_reuse_wait_desc FROM sys.databases;
```

The values that can be returned for the **log_reuse_wait_desc** column include:

0 = Nothing

1 = Checkpoint

2 = Log backup

3 = Active backup or restore

4 = Active transaction

5 = Database mirroring

6 = Replication

7 = Database snapshot creation

8 = Log scan

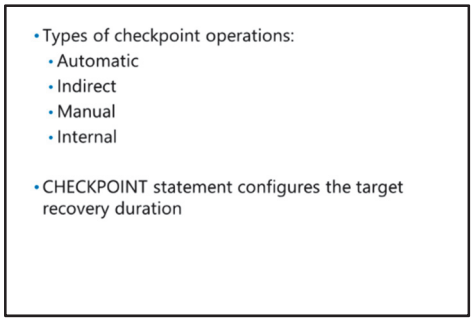
9 = Other (transient)

After resolving the reason that is shown, perform a log backup (if you are using full recovery model) to truncate the log file, and then you can use DBCC SHRINKFILE to reduce the file size of the log file.

 **Note:** If the log file does not reduce in size when using DBCC SHRINKFILE as part of the above steps, the active part of the log file must have been at the end at that point in time.

Working with Checkpoint Options

SQL Server has four types of checkpoint operation:



- Automatic checkpoints occur in the background to meet the upper time limit suggested by the recovery interval server configuration option. Automatic checkpoints run to completion, but are throttled, based on the number of outstanding writes and whether the database engine detects an increase in write latency above 20 milliseconds.
- Indirect checkpoints are issued in the background to meet a user-specified target recovery time for a given database. The default target recovery time is zero, which causes automatic checkpoint settings to be used on the database. If you have used the ALTER DATABASE statement to modify the TARGET_RECOVERY_TIME option to a value greater than zero, this value is used in place of the recovery interval specified for the server instance.
- Manual checkpoints are issued when you execute a Transact-SQL CHECKPOINT command. The manual checkpoint occurs in the current database for your connection. By default, manual checkpoints run to completion. The optional checkpoint duration parameter specifies a requested amount of time, in seconds, for the checkpoint to complete.
- Internal checkpoints are issued by various server operations, such as backup and database snapshot creation, to guarantee that disk images match the current state of the log.

You can configure the target duration of a checkpoint operation by executing the CHECKPOINT statement.

Using the CHECKPOINT Statement

```
CHECKPOINT 5;
```

Demonstration: Logs and Full Recovery

In this demonstration, you will see how log truncation works in the full recovery model.

Demonstration Steps

1. In SQL Server Management Studio, in Object Explorer, expand **Databases**, right-click **LogTest**, and then click **Properties**.
2. In the **Database Properties - LogTest** dialog box, on the **Options** page, verify that the **Recovery model** is set to **Full**, and then click **Cancel**.
3. On the **File** menu, point to **Open**, and then click **File**.
4. In the **Open File** dialog box, go to **D:\Demofiles\Mod05**, click **LogComparisonTest.sql**, and then click **Open**.
5. Select the code under the comment **Perform a full database backup**, and then click **Execute**.
6. Select the code under the comment **View log file space**, and then click **Execute**. Note the log size and space used in the **LogTest** log.

7. Select the code under the comment **Insert data**, and then click **Execute** to insert 10000 rows.
8. Select the code under the comment **View log file space**, and then click **Execute**. Note that the log size and space used in the **LogTest** log file has increased.
9. Select the code under the comment **Issue checkpoint**, and then click **Execute** to force SQL Server to perform a checkpoint and flush the modified pages to disk.
10. Select the code under the comment **View log file space**, and then click **Execute**. Note the space used in the **LogTest** log file has not decreased.
11. Select the code under the comment **Check log status**, and then click **Execute**. Note that SQL Server is awaiting a log backup before the log file can be truncated.
12. Select the code under the comment **Perform a log backup**, and then click **Execute**.
13. Select the code under the comment **Verify log file truncation**, and then click **Execute**. Note the space used in the **LogTest** log file has decreased because the log has been truncated.
14. Close SSMS without saving any changes.

Question: What are the unique features of transaction log restores?

Lesson 3
Planning Backup Strategies

To effectively plan a backup strategy, you should align your chosen combination of backup types to your business recovery requirements. Most organizations will need to use a combination of backup types rather than relying solely on just one.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand SQL Server backup types.
- Plan full database backup strategies.
- Plan transaction log backup strategies.
- Plan differential backup strategies.
- Plan partial backup strategies.

Overview of Microsoft SQL Server Backup Types

Before exploring any of the backup types in detail, it is important to be familiar with all the backup types that are available in SQL Server. Not all backup types are available for all database recovery models. For example, transaction log backups cannot be made for a database that is in simple recovery model.

Backup type	Description
Full	All data files and the active part of the transaction log
Differential	The parts of the database that have changed since the last full database backup
Partial	The primary filegroup, every read/write filegroup, and any specified read-only filegroups
Transaction Log	Any database changes recorded in the log files
Tail-log	Log backup taken of the tail of the log just before a restore operation
File/File Group	Specified files or filegroups
Copy Only	The database or log (without affecting the backup sequence)

Full Backups

A full backup of a database includes the data files and the active part of the transaction log. The first step in the backup is that a CHECKPOINT operation is performed. The active part of the transaction log includes all details from the oldest active transaction forward. A full backup represents the database at the time that the data reading phase of the backup was completed—this serves as your baseline in the event of a system failure. Full backups do not truncate the transaction log.

Differential Backups

A differential backup is used to save the data that has been changed since the last full backup. Differential backups are based on the data file contents, rather than on log file contents, and contain extents that have been modified since the last full database backup. Differential backups are generally faster to restore than transaction log backups, but they have less options available. For example, point-in-time recovery is not available unless differential backups are also combined with log file backups.

Partial Backups

A partial backup is similar to a full backup, but does not contain all of the filegroups. Partial backups contain all the data in the primary filegroup, every read/write filegroup, and any specified read-only files. A partial backup of a read-only database contains only the primary filegroup.

Note: Working with partial backups is an advanced topic that is beyond the scope of this course.

Transaction Log Backups

Transaction log backups record any database changes by backing up the log records from the transaction log. Point-in-time recovery is possible with transaction log backups, and they are generally much smaller than full database backups. The smaller size of transaction log backups means they can be run much more frequently. After the transaction log is backed up, the log records that have been backed up, and are not in the currently active portion of the transaction log, are truncated. Transaction log backups are not available in the simple recovery model.

Tail-log Backups

A transaction log backup that is taken just before a restore operation is called a tail-log backup. Typically, tail-log backups are taken after a disk failure that affects data files only. From SQL Server 2005 onwards, SQL Server has required that you take a tail-log backup before it will allow you to restore a database—to protect against inadvertent data loss.

Also, tail-log backups are often possible even when the data files from the database are no longer accessible.

File or Filegroup Backups

If performing a full database backup on very large databases is not practical, you can perform database file or filegroup backups.

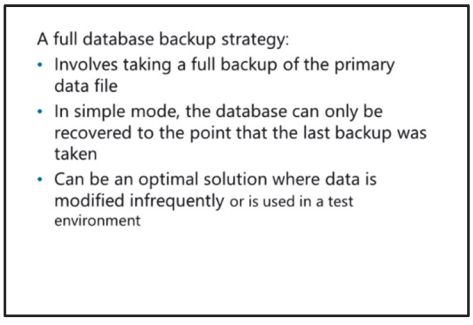
Note: Working with file and filegroup backups is an advanced topic that is beyond the scope of this course.

Copy-only Backups

SQL Server supports the creation of copy-only backups. Unlike other backups, a copy-only backup does not impact the overall backup and restore procedures for the database. Copy-only backups can be used to create a copy of the backup to take offsite to a safe location. Copy-only backups are also useful when performing some online restore operations. All recovery models support copy-only data backups.

Full Database Backup Strategies

A full database backup strategy involves regular full backups to preserve the database. If a failure occurs, the database can be restored to the state of the last full backup.



Full Database Backups

This backs up the whole database, in addition to the portion of the transaction log covering changes that occurred while reading the data pages. Full database backups represent a copy of the database at the time that the data reading phase of the backup finished, not at the time it started. Backups can be taken while the system is

being used. At the end of the backup, SQL Server writes transaction log entries, which cover the period when the backup was occurring, into the backup.

Common Usage Scenarios

For a small database that can be backed up quickly, the best practice is to use full database backups. However, as a database becomes larger, full backups take more time to complete and require more storage space. Therefore, for a large database, you might want to supplement full database backups in conjunction with other forms of backup.

After each backup, under the simple recovery model, the database is exposed to potential work loss if a disaster was to occur. The work-loss exposure increases with each update until the next full backup, when it returns to zero and a new cycle of work-loss exposure begins.

Scenarios that might be appropriate for using a full database backup strategy include:

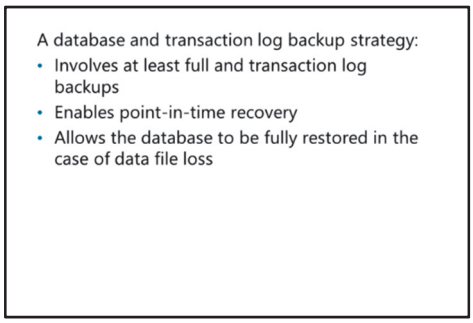
- Test systems.
- Data warehouses where the data can be recovered from a source system and where the data in the data warehouse does not change regularly.
- Systems where the data can be recovered from other sources.

Example

For example, you could perform a full backup of your database on Sunday, Monday, and Tuesday. This means that during the day on Monday, up to a full day of data is exposed to risk until the backup is performed. The same amount of exposure happens on Tuesday. After the Tuesday backup is carried out, the risk increases every day until the next Sunday backup is performed.

Transaction Log Backup Strategies

A backup strategy that involves transaction log backups must be combined with a full database strategy or a strategy that combines the use of full and differential database backups.



Transaction Log Backups

Transaction log backups save all data since the last log backup. Rather than reading database pages, transaction log backups are based on reading data from the transaction log. A backup strategy based on transaction log backups is appropriate for databases with frequent modifications.

When it is necessary to recover a database, the latest full database backup needs to be restored, along with the most recent differential backup (if one has been performed). After the database has been restored, transaction logs that have been backed up since that time are also restored, in order. Because the restore works on a transactional basis, you can restore a database to a specific point in time, within the transactions stored in the log backup.

In addition to providing capabilities that enable you to restore the transactions that have been backed up, a transaction log backup truncates the transaction log. This enables VLFs in the transaction log to be reused. If you do not back up the log frequently enough, the log files can fill up.

Example

For example, you could supplement your nightly full database backups with periodic transaction log backups during the day. If your system fails, you can recover to the time of your last transaction log backup. If, however, only the database data files failed, and a tail-log backup could be performed, no committed data loss would occur.

Combinations of Backup Types

Transaction log backups are typically much smaller than other backups, especially when they are performed regularly. Potential data loss can be minimized by a backup strategy that is based on transaction log backups, in combination with other backup types.

As log backups typically take longer to restore than other types of backup, it is often advisable to combine transaction log backups with periodic differential backups. During a recovery, only the transaction log backups that were taken after the last differential backup need to be restored.

Differential Backup Strategies

Differential backups are a good way to reduce potential work loss and maintenance overhead. When the proportion of a database that is changed between backup intervals is much smaller than the entire size of the database, a differential backup strategy can be useful. However, if you have a very small database, differential backups may not save much time.

A differential backup strategy:

- Involves performing full and differential database backups
- Includes differential backups with only changed data
- Is useful if only a subset of a database is modified more frequently than the rest of the database
- Use **modified_extent_page_count** to determine whether to perform a differential or full database backup

Differential Backups

From the time that a full backup occurs, SQL Server maintains a map of extents that have been modified. In a differential backup, SQL Server backs up only those extents that have changed. However, it is important to realize that, after the differential backup is performed, SQL Server does not clear that map of modified extents. The map is only cleared when full backups occur. This means that a second differential backup performed on a database will include all changes since the last full backup, not just those changes since the last differential backup.

Common Usage Scenarios

Because they only save the data that has been changed since the last full database backup, differential backups are typically much faster and occupy less disk space than transaction log backups for the same period of time.

Differential database backups are especially useful when a subset of a database is modified more frequently than the remainder of the database. In these situations, differential database backups enable you to back up frequently, without the overhead of full database backups.

For example, you might take full database backup at midnight on Sunday (early Monday morning). You could then take differential backups at midnight each other night of the week. The differential backup taken on Monday night would include all data changed during Monday. The differential backup taken on Tuesday night would include all data changed on Monday and Tuesday. The differential backup taken on Friday night would include all data that changed on Monday, Tuesday, Wednesday, Thursday, and Friday. This means that differential backups can grow substantially in size between each full backup interval.

Combinations of Backups

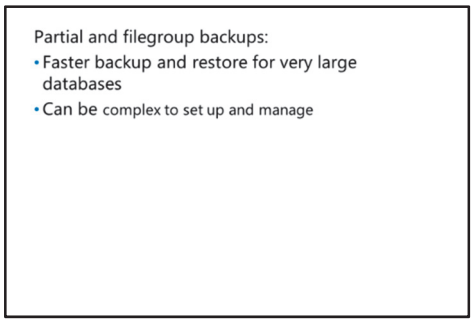
Differential backups must be combined with other forms of backup. Because a differential backup saves all data changed since the last full backup was made, it cannot be taken unless a full backup has already been performed.

Another important aspect to consider is that, when a recovery is required, multiple backups need to be restored to bring the system back online—rather than a single backup. This increases the risk exposure for an organization and must be considered when planning a backup strategy.

Differential backups can also be used in combination with both full and transaction log backups.

SQL Server 2017 introduces a new column, **modified_extent_page_count**, in the **sys.dm_db_file_space_usage** table. This column returns the number of extents that have changed since the last full back up. You can use this and the **total_page_count** column to calculate the percentage of extents that have changed to decide whether to perform a differential or full database backup.

Partial and Filegroup Backup Strategies



For very large databases that contain multiple data files and/or multiple filegroups, you can consider using a file or filegroup backup strategy to reduce the time it takes to perform backups. This strategy is useful for databases that have narrow backup windows, and it can also speed up recovery times—if a single data file is lost or corrupt, you only need to restore that file or the filegroup that contains that file, instead of the whole database. As with the other backup strategies, you initiate this by taking a full database backup, but you then back up the data files or filegroups in turn. You can also back up transaction logs between file or filegroup backups, to improve recoverability.

Managing file and filegroup backups can be complex, and the loss of a single data file backup can cause serious problems, including making a database unrecoverable.

One way to simplify the process of backing up parts of a database is to use a partial backup, which backs up only the primary filegroup and the read/write filegroups. However, this is only recommended when the database contains enough data in read-only filegroups to make a substantial time and administrative saving. It is also recommended that you use partial backups in conjunction with the simple recovery model.

One of the key benefits of a partial backup strategy is that, in the event of a failure, you can perform a piecemeal restore that makes data in the read/write filegroups available before the read-only filegroups have been restored. This means you can reduce the recovery time for workloads that do not require the data in the read-only filegroups.

For example: all read-only filegroups are backed up at midnight on Monday, along with a partial backup that includes only the primary filegroup and all read/write filegroups. At the end of each subsequent day, a partial differential backup is used to back up modified pages in read/write filegroups.

Question: What kind of database might be a good candidate for a full backup strategy?

Lab: Understanding SQL Server Recovery Models

Scenario

You need to implement a database recovery strategy. The business unit from Proseworks Inc. has provided the availability needs for the databases on the new Proseware SQL Server instance. You need to plan how you will meet the requirements, and then implement your strategy.

If you have time, there is another issue that your manager would like you to work on. There is another instance of SQL Server installed for supporting customer service operations. Your manager is concerned that existing databases on the CustomerService server instance are configured inappropriately and have invalid backup strategies, based on their RPO and RTO requirements. You need to review the database recovery models and backup strategies for the databases on the CustomerService instance, and provide recommended changes.

Supporting Documentation

Business Database Continuity Requirements for Databases on the Proseware Server Instance (for Exercises 1 and 2):

- **Recovery Time Objectives**
  - The MarketDev database must never be unavailable for longer than eight hours.
  - The Research database must never be unavailable for longer than two hours.
- **Recovery Point Objectives**
  - When the MarketDev database is recovered from a failure, no more than 30 minutes of transactions may be lost.
  - When the Research database is recovered from a failure, all transactions that were completed up to the end of the previous weekday must be recovered.

Projected Characteristics

Characteristic	Estimated Value
MarketDev database size	20 GB
Research database size	200 MB
Total backup throughput	100 MB/minute
Total restore throughput	80 MB/minute
Average rate of change to the MarketDev database during office hours	1 GB/hour
Average rate of change to the Research database during office hours	10 MB/hour
Percentage of the MarketDev database changed each day (average)	1.2%
Percentage of the Research database changed each day (average)	80%
Office hours (no full database backups permitted during these hours)	08:00 to 18:00

Business Database Continuity Requirements for Databases on the CustomerService Server Instance (for Exercise 3):

- **Recovery Time Objectives**
  - The CreditControl database must never be unavailable for longer than two hours.
  - The PotentialIssue database must never be unavailable for longer than one hour.
- **Recovery Point Objectives**
  - When the CreditControl database is recovered from a failure, no more than five minutes of transactions may be lost.
  - When the PotentialIssue database is recovered from a failure, no more than 30 minutes of transactions may be lost.

Projected Characteristics

Characteristic	Estimated Value
CreditControl database size	20 GB
PotentialIssue database size (at the start of each week after archiving activity is complete)	200 MB
Total backup throughput	100 MB/minute
Total restore throughput	80 MB/minute
Average rate of change to the CreditControl database during office hours	500 MB/hour
Average rate of change to the PotentialIssue database (constant throughout the week, 24 hours per day)	10 MB/hour
Percentage of the CreditControl database changed each day (average)	60%
Percentage of the PotentialIssue database changed each day (average)	50%
Office hours (no full database activity permitted during these hours)	08:00 to 19:00

Existing Backup Strategy for CreditControl Database

Recovery Model: Full

Type of Backup	Schedule
Full	Saturday 06:00 Wednesday 06:00
Differential	Sunday 22:00 Monday 22:00 Tuesday 22:00 Thursday 22:00 Friday 22:00
Transaction Log	Every 60 minutes on the hour

Existing Backup Strategy for PotentialIssue Database

Recovery Model: Full

Type of Backup	Schedule
Full	Sunday 22:00
Log	Every 15 minutes, starting at 10 minutes past the hour

Objectives

After completing this lab, you will be able to:

- Plan a backup strategy.
- Configure database recovery models.
- Check that a chosen recovery model meets specific requirements.

Estimated Time: 45 minutes

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

Username: **AdventureWorks\Student**

Password: **Pa55w.rd**

Exercise 1: Plan a Backup Strategy

Scenario

You need to plan a backup strategy for the two databases on the new Proseware instance. You have been provided with RPO and RTO for both databases, as part of a business continuity statement.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Review the Business Requirements

3. Determine an Appropriate Backup Strategy

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab05\Starter** folder as Administrator.

▶ Task 2: Review the Business Requirements

- Review the supplied business requirements in the supporting documentation for this exercise.

▶ Task 3: Determine an Appropriate Backup Strategy

- Determine an appropriate backup strategy for each database including:
  - The recovery model that should be used.
  - The type of backup.
  - The backup schedule.

 **Note:** To meet the business requirements, each database might need more than one backup type and schedule.

Results: At the end of this exercise, you will have created a plan to back up two databases.

Exercise 2: Configure Database Recovery Models

Scenario

You have carried out a review to identify the appropriate database recovery models for the needs of the business. In this exercise, you need to set the recovery models for the databases that do not meet the requirements.

The main tasks for this exercise are as follows:

1. Review and Adjust Database Recovery Models

▶ Task 1: Review and Adjust Database Recovery Models

- Review the recovery models that you decided were required in Exercise 1; check whether or not the existing recovery models for the **MarketDev** and **Research** databases match your recommendations. If not, use SSMS to change the recovery models as per your recommendations.

Results: At the end of this exercise, you will have modified the database recovery models where required.

Exercise 3: Challenge Exercise: Review Recovery Models and Strategy (if time permits)

Scenario

There is another instance of SQL Server installed for supporting customer service operations named MIASQL\SQL2. There is concern that existing databases on the SQL2 server instance are configured inappropriately and have invalid backup strategies, based on their RPO and RTO requirements. In this exercise, you need to review the database recovery models and backup strategies for the databases on the SQL2 instance and provide recommended changes.

The main tasks for this exercise are as follows:

1. Review the RPO and RTO Requirements for the Databases

2. Review the Existing Recovery Models and Backup Strategies

3. Indicate Whether or Not the Strategy Would Be Successful

▶ Task 1: Review the RPO and RTO Requirements for the Databases

- The supporting documentation includes details of the business continuity requirements for the databases. Review this documentation.

▶ Task 2: Review the Existing Recovery Models and Backup Strategies

- The supporting documentation also includes details of the backup strategy for the databases. Review this documentation.

▶ Task 3: Indicate Whether or Not the Strategy Would Be Successful

- Assess whether or not the current backup strategy and recovery model configuration is capable of supporting the business continuity requirements. If not, explain why it would not work.

Results: At the end of this exercise, you will have assessed the backup strategy.

Module Review and Takeaways

In this module, you have learned how to create a backup strategy that is aligned with organizational needs. You have also seen how the transaction logging capabilities of SQL Server can help you to achieve an appropriate outcome.

Best Practice:

- Plan your backup strategy carefully.
- Plan the backup strategy in conjunction with the business needs.
- Choose the appropriate database recovery model.
- Plan your transaction log size, based on the transaction log backup frequency.
- Consider using differential, partial, and filegroup backups to speed recovery.

Review Question(s)

Question: When might a full database backup strategy be adequate?

Module 6

Backing Up SQL Server Databases

Contents:

Module Overview

In the previous module, you learned how to plan a backup strategy for a SQL Server® system. You can now learn how to perform SQL Server backups, including full and differential database backups, transaction log backups, and partial backups.

In this module, you will learn how to apply various backup strategies.

Objectives

After completing this module, you will be able to:

- Perform backups of SQL Server databases and transaction logs.
- Manage database backups.
- Describe advanced backup options.

Lesson 1
Backing Up Databases and Transaction Logs

Now you have seen how to plan a backup strategy for a SQL Server system, you can learn how to perform SQL Server backups, including full and differential database backups, transaction log backups, and partial backups.

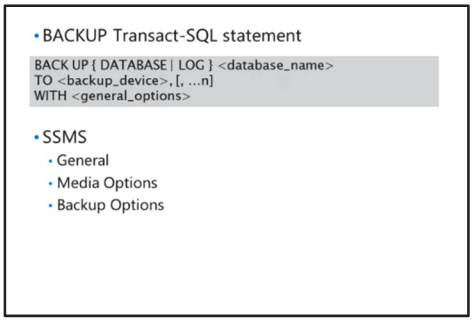
Lesson Objectives

After completing this lesson, you will be able to:

- Describe media sets and backup sets.
- Back up SQL Server databases.
- Back up transaction logs.
- Perform partial and filegroup backups.

Introduction to SQL Server Backups

You can perform backups in SQL Server by using the BACKUP Transact-SQL statement or the graphical interface in SQL Server Management Studio (SSMS).



Note: You can also use Windows PowerShell® to perform a backup by using the **Backup-SqlDatabase** cmdlet in the **SQLPS** module.

Simplified syntax for the BACKUP statement is shown below:

Simplified BACKUP Syntax

```
BACKUP { DATABASE | LOG } <database_name>
TO <backup_device>, [,…n]
WITH <general_options>
```

The SSMS graphical user interface includes the following pages, on which you can configure backup options:

- **General**. Use this page to specify the database to be backed up, the backup type, the backup destination, and other general settings.
- **Media Options**. Use this page to control how the backup is written to the backup device(s); for example, overwriting or appending to existing backups.
- **Backup Options**. Use this page to configure backup expiration, compression, and encryption.

In SQL Server, you perform backups while other users continue working with the database; and these other users might experience a performance impact due to the I/O load placed on the system by the backup operation. SQL Server does place some limitations on the types of commands you can execute while a backup is being performed. For example, you cannot use the **ALTER DATABASE** command with

the **ADD FILE** or **REMOVE FILE** options or shrink a database during a backup. Additionally, you cannot include the **BACKUP** command in either an explicit or an implicit transaction, or to roll back a backup statement.

You can only back up databases when they are online, but it is possible to perform a backup of the transaction log when a database is damaged, if the log file itself is still intact. This is why it is so important to split data and log files onto separate physical media.

Backup Timing

An important consideration when making a backup is to understand the timing associated with its contents—the database may be in use while the backup is occurring. For example, if a backup starts at 22:00 and finishes at 01:00, does it contain a copy of the database as it was at 22:00, a copy as it was at 01:00, or a copy from a time between the start and finish?

SQL Server writes all data pages to the backup device in sequence, but uses the transaction log to track any pages that are modified while the backup is occurring. SQL Server then writes the relevant portion of the transaction log to the end of the backup. This process makes the backups slightly larger than in earlier versions, particularly if heavy update activities are happening at the same time. This altered process also means that the backup contains a copy of the database as it was at a time just before the completion of the backup—not as it was at the time the backup started.

VSS and VDI

The Windows Volume Shadow Copy Service (VSS) and the Virtual Device Interface (VDI) programming interfaces are available for use with SQL Server. This is so that third-party backup tools can work with SQL Server.

In very large systems, it is common to have to perform disk-to-disk imaging while the system is in operation, because standard SQL Server backups might take too long to be effective. With the VDI programming interface, an application can freeze SQL Server operations momentarily while it takes a consistent snapshot of the database files. This form of snapshot is commonly used in geographically-distributed storage area network (SAN) replication systems.

Media Sets and Backup Sets

Before performing a backup, you should understand how backups are stored. A single backup is called a backup set, and is written to a media set, which can contain up to 64 backup devices.

- Media sets consist of one or more disk backup devices
  - Data is striped across multiple devices
- A backup set represents one backup of any type
- Backup sets are written to media sets
  - A media set can contain multiple backup sets
- Backup devices and media sets are created at first use:
  - Use FORMAT to overwrite an existing media set
  - Use INIT to overwrite existing backup sets in a media set
  - Use the FORMAT option with caution

Backup devices can be physical or logical. Physical backup devices are explicit files that you specify using a Universal Naming Convention (UNC) file path; logical devices are named objects that reference files, providing a layer of abstraction that makes it easier to change backup device locations. Ultimately, both physical and logical devices reference a file location, which can be on the local file system, a network file share, a volume in a SAN or other storage device, or a blob container in Microsoft Azure® storage.

Note: Direct backup to tape is not supported. If you want to store backups on tape, you should first write it to disk, and then copy the disk backup to tape.

If a media set spans several backup devices, the backups will be striped across the devices.

 **Note:** No parity device is used while striping. If two backup devices are used together, each receives half the backup. Both must also be present when attempting to restore the backup.

Every backup operation to a media set must write to the same number and type of backup devices. Media sets and the backup devices are created the first time a backup is attempted on them. Media sets and backup sets can also be named at the time of creation and given a description.

The backups on an individual device within a media set are referred to as a media family. The number of backup devices used for the media set determines the number of media families in a media set. For example, if a media set uses two backup devices, it contains two media families.

Media and Backup Set Initialization

Media sets can contain multiple backup sets, so that you can append backups of different types (and even different databases) to the same backup devices. When you perform a backup, you must be careful to use the appropriate media set options to avoid inadvertently overwriting backup sets that you want to retain.

Two key options of which you must be particularly aware are:

- **FORMAT**/**NOFORMAT**. The **FORMAT** option is used to write a new media header on the backup devices used in the backup. This creates a new media set, breaking any existing media sets to which the backup devices currently belong, and deleting any existing backup sets they contain. When you perform a backup to an existing backup device, SQL Server uses a default value of **NOFORMAT** to safeguard against accidental backup deletion. In SQL Server Management Studio, the **FORMAT** option is specified by selecting **Back up to a new media set, and erase all existing backup sets** in the **Backup Database** dialog box.
- **INIT**/**NOINIT**. The **INIT** option retains the existing media header, but overwrites all existing backup sets in the media set. By default, SQL Server uses the **NOINIT** option to avoid accidental backup deletion. In SQL Server Management Studio, you can select **Backup to the existing media set**, and then select **Append to the existing backup set** to use the **NOINIT** option, or **Overwrite all existing backups** to use the **INIT** option.

As an example, consider the following code, which backs up a database to a media set that consists of two files. Assuming the files do not already exist, SQL Server creates them and uses them to define a new media set. The data from the backup is striped across the two files:

Initializing a Media Set

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\SQLBackups\AW_1.bak',
   DISK = 'E:\SQLBackups\AW_2.bak';
```

Another backup could be made later, to the same media set. The data from the second backup is again striped across the two files, and the header of the media set is updated to indicate that it now contains the two backups:

Appending a Backup Set to an Existing Media Set

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\SQLBackups\AW_1.bak',
   DISK = 'E:\SQLBackups\AW_2.bak'
WITH NOINIT;
```

If a user then tries to create another backup to only one of the backup files in the media set using the following code, SQL Server will return an error, because all backup sets to a media set must use the same backup devices:

Attempting to Use a Single Device in an Existing Media Set

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\SQLBackups\AW_1.bak';
```

Before the member of the media set can be overwritten, the FORMAT option has to be added to the WITH clause in the backup command. This creates a new media set that contains a single file. The original media set, together with all of the backup sets it contains, is no longer valid:

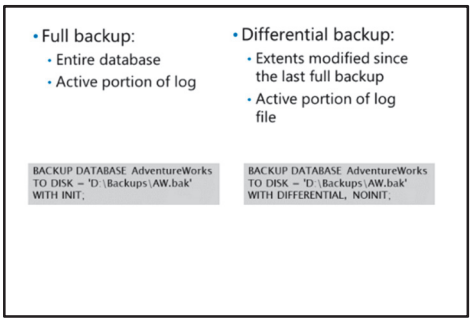
Formatting a Media Set

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\SQLBackups\AW_1.bak'
WITH FORMAT, INIT;
```

Use the FORMAT option to overwrite the contents of a backup file and split up the media set, but use it very carefully. Formatting one backup file of a media set renders the entire backup set unusable.

Performing Database Backups

Database backups can be full or differential.



Performing a Full Database Backup

A full database backup will back up all of the data pages in the database and also saves the active portion of the transaction log.

The following code makes a full database backup of the AdventureWorks database and stores it in a file named 'D:\Backups\AW.bak'. The INIT option creates the file if it does not already exist and overwrites it if it does:

Performing a Full Database Backup

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW.bak'
WITH INIT;
```

Performing a Differential Database Backup

Although full database backups are ideal, often the time taken to perform one can outweigh the benefits that it provides. This is particularly true when only a small percentage of the database changes between each backup. In this scenario, differential backups are a sensible consideration. You can perform a differential backup by using SSMS or by using the DIFFERENTIAL option of the BACKUP statement.

The following code makes a differential database backup of the AdventureWorks database and stores it in a file named 'D:\Backups\AW.bak'. The NOINIT option appends the backup to any existing backups in the media set:

Performing a Differential Database Backup

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW.bak'
WITH DIFFERENTIAL, NOINIT;
```

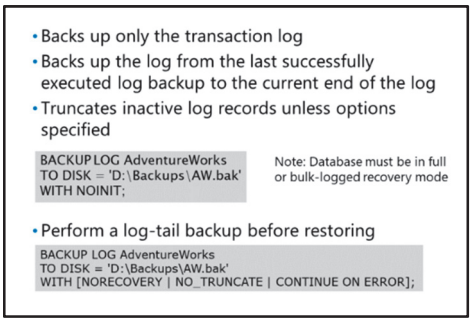
How Differential Backups Work

SQL Server maintains a map of modified extents called the differential bitmap page. One differential bitmap page is maintained for each 4-GB section of every data file. Each time a full database backup is created, SQL Server clears the map. As the data in the data files changes, SQL Server updates the map and, when a differential backup runs, only extents that have changed since the last full database backup are backed up. If you run two consecutive differential backups, the second will contain all the extents that have changed since the last full database backup, not just the ones since the first differential backup.

Note: You cannot create a differential database backup unless a full database backup has been taken first.

Performing Transaction Log Backups

Before a transaction log backup can be performed, the database must be in either **full** or **bulk-logged** recovery mode. In addition, a transaction log backup can only occur when a full database backup has been previously taken. Unless a database is set to **bulk-logged** recovery mode, a transaction log backup does not save any data pages from the database.



The following code backs up the transaction log of the **AdventureWorks** database:

Performing a Transaction Log Backup

```
BACKUP LOG AdventureWorks
TO DISK = 'D:\Backups\AW.bak'
WITH NOINIT;
```

A transaction log backup finds the **MaxLSN** of the last successful transaction log backup, and saves all log entries beyond that point to the current MaxLSN. The process then truncates the transaction log as far as is possible (unless the **COPY_ONLY** or **NO_TRUNCATE** option is specified). The longest-running active transaction must be retained, in case the database has to be recovered after a failure.

Log Record Chains

Before you can restore a database by using transaction log backups, an unbroken chain of log records must be available—from the last full database backup to the desired point of restoration. If there is a break, you can only restore up to the point where the backup chain is broken.

For example, imagine a scenario where you create a database, and later take a full backup. At this point, the database can be recovered. If the recovery model of the database is then changed to **simple** and

subsequently switched back to **full**, a break in the log file chain has occurred. Even though a previous full database backup exists, the database can only be recovered up to the point of the last transaction log backup, before the change to simple recovery mode.

After switching from simple to full recovery model, you must perform a full database backup to create a starting point for transaction log backups.

Backing Up the Tail-log

To recover a database to the point of failure, you must take a tail-log backup before starting a restore on an existing database. This ensures that all transactions are written to at least one backup, before they can be overwritten. The tail-log backup prevents work loss and keeps the log chain intact. When you are recovering a database to the point of a failure, the tail-log backup is often the last one of interest in the recovery plan. If you cannot back up the tail of the log, you can only recover a database to the end of the last backup that was created before the failure.

Note: Not all restore scenarios require a tail-log backup. You do not need to have a tail-log backup if the recovery point is contained in an earlier log backup or if you are moving or replacing (overwriting) the database and do not have to restore it to a point of time after the most recent backup.

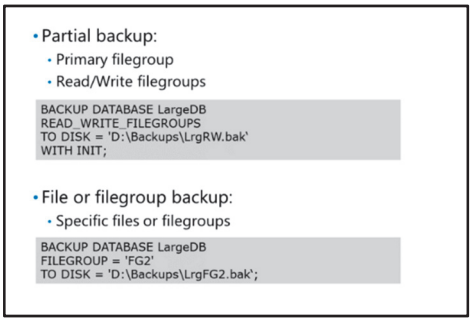
When performing a tail-log backup of a database that is currently online, you can use the NO_RECOVERY option to immediately place the database into a restoring state, preventing any more transactions from occurring until the database is restored.

If the database is damaged, you can use the NO_TRUNCATE option, which causes the database engine to attempt the backup, regardless of the state of the database. This means that a backup taken while using the NO_TRUNCATE option might have incomplete metadata.

If you are unable to back up the tail of the log using the NO_TRUNCATE option when the database is damaged, you can attempt a tail-log backup by specifying the CONTINUE_AFTER_ERROR option.

Performing Partial and Filegroup Backups

When managing an extremely large database, the time taken to perform full backups (and restore them in the event of failure) can have a detrimental effect on ongoing business operations. While transaction log backups and differential backups can ease this problem, for extremely large databases that use the **Simple** recovery model (for which transaction log backups cannot be performed), you can choose to back up only the files and filegroups that contain volatile data, without including read-only files and filegroups in the backup.



There are two techniques used to implement this kind of backup solution:

- **Partial backup**. A partial backup backs up only the primary filegroup and filegroups that are set to read-write. You can also include specific read-only filegroups if required. The purpose of a partial backup is to make it easy for you to back up the parts of a database that change, without having to plan the backup of specific files or filegroups. You can perform a full or differential partial backup.

- **File and Filegroup backups**. With a filegroup backup, you can back up only selected files and filegroups in a database. This can be useful with very large databases that would take a long time to back up in full, because you have to back it up in phases. It is also useful for databases that contain some read-only data, or data that changes at different rates, because you can back up only the read-write data, or back up frequently updated data more often.

The following code example performs a partial backup that includes the primary filegroup and all read/write filegroups:

A Partial Backup

```
BACKUP DATABASE LargeDB
READ_WRITE_FILEGROUPS
TO DISK = 'D:\Backups\LrgRW.bak'
WITH INIT;
```

The following code example backs up specific filegroups. You can also use the FILE parameter to back up individual files:

A Filegroup Backup

```
BACKUP DATABASE LargeDB
FILEGROUP = 'LrgFG2'
TO DISK = 'D:\Backups\LrgFG2.bak'
WITH INIT;
```

Demonstration: Performing Backups

In this demonstration, you will see how to:

- Perform a full database backup.
- Perform a differential database backup.
- Perform a transaction log backup.

Demonstration Steps

Perform a Full Database Backup

1. Ensure that you have started the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines, log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod06** folder, run **Setup.cmd** as Administrator. Wait for the script to finish, and then press enter.
3. Start SQL Server Management Studio, and connect to the **MIA-SQL** database engine using Windows authentication.
4. In Object Explorer, expand **Databases**, right-click **AdventureWorks**, point to **Tasks**, and click **Back Up**.
5. In the **Back Up Database - AdventureWorks** dialog box, ensure that **Backup type** is set to **Full**.
6. In the **Destination** section, select each existing file path and click **Remove**, and then click **Add**.
7. In the **Select Backup Destination** dialog box, in the **File name** box, type **D:\Demofiles\Mod06\Demo\AW.bak**, and then click **OK**.

8. In the **Back Up Database - AdventureWorks** dialog box, on the **Media Options** page, note that the default option is to append to an existing media set. In this case, there is no existing media set so a new one will be created, and there are no existing backup sets to overwrite.
9. In the **Back Up Database - AdventureWorks** dialog box, on the **Backup Options** page, note the default backup name and expiration settings.
10. In the **Back Up Database - AdventureWorks** dialog box, in the **Script** drop-down list, click **Script Action to New Query Window**, and then click **OK**.
11. When the backup has completed successfully, click **OK**.
12. In the query pane, view the Transact-SQL BACKUP statement that was used to back up the database.
13. View the **D:\Demofiles\Mod06\Demo** folder and note the size of the **AW.bak** file.

Perform a Differential Backup

1. In SQL Server Management Studio, open the **UpdatePrices.sql** script file from the **D:\Demofiles\Mod06\Demo** folder, and click **Execute**. This script updates the **Production.Product** table in the **AdventureWorks** database.
2. In Object Explorer, under **Databases**, right-click **AdventureWorks**, point to **Tasks**, and click **Back Up**.
3. In the **Back Up Database - AdventureWorks** dialog box, in the **Backup type** list, click **Differential**.
4. In the **Destination** section, ensure that **D:\Demofiles\Mod06\Demo\AW.bak** is the only backup device listed.
5. In the **Back Up Database - AdventureWorks** dialog box, on the **Media Options** page, verify that the option to append to the existing media set is selected.
6. In the **Back Up Database - AdventureWorks** dialog box, on the **Backup Options** page, change the **Name** to **AdventureWorks-Diff Database Backup**.
7. In the **Back Up Database - AdventureWorks** dialog box, in the **Script** drop-down list, click **Script Action to New Query Window**, and then click **OK**.
8. When the backup has completed successfully, click **OK**.
9. In the query pane, view the Transact-SQL BACKUP statement that was used to back up the database. Note that it includes the WITH DIFFERENTIAL option.
10. View the **D:\Demofiles\Mod06\Demo** folder, and note that the size of the **AW.bak** file has increased, but not much—the second backup only includes the extents containing pages that were modified since the full backup.

Perform a Transaction Log Backup

1. In SQL Server Management Studio, switch to the **UpdatePrices.sql** script you opened previously, and click **Execute** to update the **Production.Product** table in the **AdventureWorks** database again.
2. In Object Explorer, under **Databases**, right-click **AdventureWorks,** point to **Tasks**, and click **Back Up**.
3. In the **Back Up Database - AdventureWorks** dialog box, in the **Backup type** list, click **Transaction Log**.
4. In the **Destination** section, ensure that **D:\Demofiles\Mod06\Demo\AW.bak** is the only backup device listed.

5. In the **Back Up Database - AdventureWorks** dialog box, on the **Media Options** page, verify that the option to append to the existing media set is selected. Also verify that the option to truncate the transaction log is selected.
6. In the **Back Up Database - AdventureWorks** dialog box, on the **Backup Options** page, change the **Name** to **AdventureWorks-Transaction Log Backup**.
7. In the **Back Up Database - AdventureWorks** dialog box, in the **Script** drop-down list, click **Script Action to New Query Window**, and then click **OK**.
8. When the backup has completed successfully, click **OK**.
9. In the query pane, view the Transact-SQL BACKUP statement that was used to back up the database. Note that this time the statement is BACKUP LOG.
10. View the **D:\Demofiles\Mod06** folder and note that the size of the **AW.bak** file has increased, but not much—the third backup only includes the transaction log entries for data modifications since the full backup.
11. Keep SQL Server Management Studio open for the next demonstration.

Lesson 2
Managing Database Backups

No matter how many backups you perform, it is essential that you make sure they are readable and restorable, otherwise the entire backup system is flawed. It is also important to be able to query information about your backups so you can access the correct data when required.

In this lesson, you will learn how to verify a backup and ensure its integrity, and how to retrieve backup history and header information.

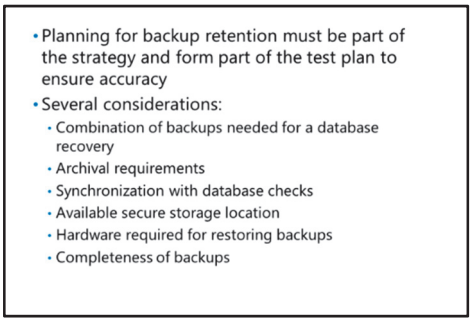
Lesson Objectives

After completing this lesson, you will be able to:

- Understand retention and testing policies.
- Describe options for backup integrity.
- View backup history.
- Retrieve backup metadata.

Determining a Retention and Testing Policy for Backups

It is too common for organizations to regularly perform backup operations, and then when the time comes to restore the backups, find that they are not usable. Most of these problems would be alleviated by a good retention and testing plan. Your strategy must include plans for the retention of backups and for the locations where the media or backups should be retained.



Several common problems arise, but these can easily be avoided.

- **Insufficient Copies of Backups**. Your organization is dependent on the quality of backups should they need to be restored. The more copies of backups you have and the more pieces of media that are holding all the required data, the better the chance you have of being able to recover them.

  The worst option is generally regarded as creating a backup over your most recent backup. If the system fails during the backup, you will often then have lost both your data and your backup.

  Avoidance strategy: make multiple copies of backups.

- **Insufficient Data on the Backups**. Company A performed regular backups, yet no testing of recovery was ever made. The first time a real recovery was attempted, it was discovered that not all files that needed to be backed up were in fact being backed up.

  Avoidance strategy: regular reconstruction of data from backup recovery testing.

- **Unreadable Backups**. Company B performed regular backups but did not test them. When recovery was attempted, none of the backups were readable. This is often initiated by hardware failures but can be caused by inappropriate storage of media.

  Avoidance strategy: regular backup recovery testing and use of redundant backup hardware.

- **Unavailable Hardware**. Company C purchased a special tape drive to perform backups. When the time came to restore the backups, that special device no longer worked, and the organization had no other way to read the backups, even if they were valid.

  Avoidance strategy: regular backup recovery testing.

- **Old Hardware**. Company D performed regular backups and retained them for an appropriate period. When the time came to restore the backups, the company no longer possessed equipment that was capable of performing that operation.

  Avoidance strategy: regular backup recovery testing, combined with recovery and backup onto current devices.

- **Misaligned Hardware**. Company E performed regular backups and even tested that they could undertake restore operations from the backups. However, because they tested the restores on the same device that performed the backups, they did not realize that the device was misaligned and it was the only one that could read those backups. When a restore was needed, the device that the backups were performed on had failed.

  Avoidance strategy: regular backup recovery testing on a separate system and physical device.

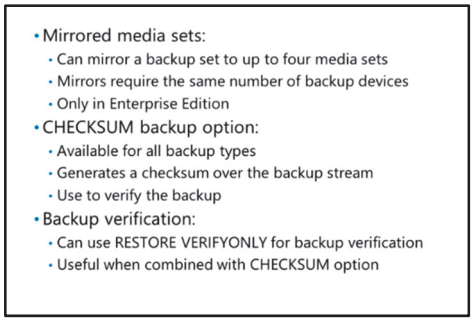
General Considerations

There are several general points to consider regarding the retention period of backups.

- When a backup strategy calls for you to perform multiple types of backups, it is important to work out the combination of backups you will require.
- Organizations might have to fulfill legal or compliance requirements regarding the retention of backups. In most cases, full database backups are kept for a longer time than other types.
- Checking the consistency of databases by using the DBCC CHECKDB statement is a crucial part of database maintenance, and is discussed later in the course.
- In addition to deciding how long backups should be retained, you must determine where they are kept. An important part of meeting the RTO is to consider how long it takes to obtain the physical backup media if it has to be restored.
- You should also make sure that backups are complete. Are all files that are needed to recover the system (including external operating system files) being backed up?

Options for Ensuring Backup Integrity

SQL Server Backup includes options to help ensure the integrity of backups, reducing the risk of finding that backups are unusable when required to recover from a failure.



Mirrored Media Sets

A mirrored media set is a copy of the backup media set that you can optionally create in parallel with the main backup operation. It consists of two to four device mirrors, each containing the entire media set. You should configure each mirror with the same number of backup devices, which must be of the same device type.

Using the theory that it is better to have multiple copies of a backup, rather than a single copy, mirroring a media set can increase availability of your data. However, it is important to realize that mirroring a media set exposes your system to a higher level of hardware failure risk, because a malfunction of any of the backup devices causes the entire backup operation to fail.

You can create a mirrored backup set by using the MIRROR TO option of the BACKUP statement:

Creating a Mirrored Backup Set

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW.bak'
MIRROR TO DISK = 'E:\ Backups\AW_M.bak'
WITH FORMAT, INIT;
```

Note: The mirrored media set functionality is only available in SQL Server Enterprise Edition.

WITH CHECKSUM Option

With SQL Server, you can perform a checksum operation over an entire backup stream, and write the value to the end of the backup. The WITH CHECKSUM option validates the page-level information (checksum or torn page if either is present) in addition to the one checksum for the backup stream. It does, however, consume slightly more CPU resources during the backup process than backups without the calculation of a checksum.

You can configure SQL Server to assess the checksum value, either during restore operations or during backup verification operations made with the RESTORE VERIFYONLY command.

You switch on the checksum option by using the WITH CHECKSUM clause of the BACKUP statement:

Using a Checksum

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW.bak'
WITH CHECKSUM;
```

Backup Verification

To verify a backup, you can use the RESTORE VERIFYONLY statement that checks the backup for validity but does not restore it. The statement performs the following checks:

- Backup set is complete.
- All volumes are readable.
- Page identifiers are correct (to the same level as if it were about to write the data).
- Checksum is valid (if present on the media).
- Sufficient space exists on destination devices.

The checksum value can only be validated if the backup was performed with the WITH CHECKSUM option. Without the CHECKSUM option during backup, the verification options only check the metadata and not the actual backup data.

The RESTORE VERIFYONLY statement is similar to the RESTORE statement and supports a subset of its arguments:

Verifying a Backup

```
RESTORE VERIFYONLY
FROM DISK = 'D:\Backups\AW.bak'
```

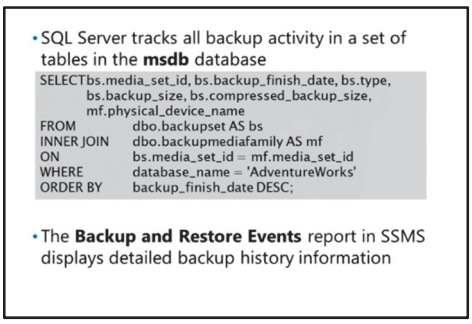
You can also perform verification steps by using the backup database task in SSMS.

Note: Consider verifying backups on a different system to the one where the backup was performed. This will eliminate the situation where a backup is only readable on the source hardware.

Viewing Backup History

SQL Server tracks all backup activity in the following tables in the **msdb** database:

- backupfile
- backupfilegroup
- backupmediafamily
- backupmediaset
- backupset



You can query these tables to retrieve information about backups that have been performed:

Querying Backup History Tables

```
SELECT       bs.media_set_id, bs.backup_finish_date, bs.type,
             bs.backup_size, bs.compressed_backup_size,
             mf.physical_device_name
FROM         dbo.backupset AS bs
INNER JOIN   dbo.backupmediafamily AS mf
ON           bs.media_set_id = mf.media_set_id
WHERE        database_name = 'AdventureWorks'
ORDER BY     backup_finish_date DESC;
```

SSMS also provides reports and logs of backup information.

You can use system stored procedures to delete backup history:

Deleting Backup History

```
// Delete all backup history before 2009
EXEC sp_delete_backuphistory @oldest_date = '20090101';

// Delete all backup history for the Market database
EXEC sp_delete_database_backuphistory @database_name = 'Sales';
```

Note: If a database is restored onto another server, the backup information is not restored with the database, because it is held in the **msdb** database of the original system.

Retrieving Backup Metadata

You can find information about a specific media set by executing the RESTORE statement with the specific options:

Option	Description
RESTORE LABELONLY	Returns information about the backup media on a specified backup device.
RESTORE HEADERONLY	Returns all the backup header information for all backup sets on a particular backup device.
RESTORE FILELISTONLY	Returns a list of data and log files contained in a backup set.

- RESTORE LABELONLY returns information about the backup media on a specified backup device
- RESTORE HEADERONLY returns all the backup header information for all backup sets on a particular backup device
- RESTORE FILELISTONLY returns a list of data and log files contained in a backup set

Demonstration: Verifying Backups

In this demonstration, you will see how to:

- View the Backup and Restore Events report.
- Query backup history tables.
- Verify backup media.

Demonstration Steps

View the Backup and Restore Events Report

1. In SQL Server Management Studio, in Object Explorer, under **Databases**, right-click **AdventureWorks**, point to **Reports**, point to **Standard Reports**, and then click **Backup and Restore Events**.
2. In the **Backup and Restore Events [AdventureWorks]** report, expand **Successful Backup Operations** and view the backup operations that have been performed for this database.
3. In the **Device Type** column, expand each of the **Disk (temporary)** entries to view details of the backup media set files.

Query Backup History Tables

1. In SQL Server Management Studio, open the **VerifyingBackups.sql** script file in the **D:\Demofiles\Mod06\Demo** folder.
2. Highlight the code under the comment **View backup history**, and click **Execute**.
3. View the query results, which show the backups that have been performed for the **AdventureWorks** database.
4. Note line 21 of the code restricts the retrieval period currently set to 30 days. This can be modified by changing the number 30 or by removing this line.

Verify Backup Media

1. Highlight the code under the comment **Use RESTORE HEADERONLY**, and click **Execute**.
2. View the query results, which show the backups in the **AW.bak** backup device.
3. Highlight the code under the comment **Use RESTORE FILELISTONLY**, and click **Execute**.
4. View the query results, which show the database files contained in the backups.
5. Highlight the code under the comment **Use RESTORE VERIFYONLY**, and click **Execute**.
6. View the message that is returned, which should indicate that the backup is valid.

Lesson 3
Advanced Database Options

SQL Server Backup provides a range of options that can help optimize your backup strategy, including the ability to perform a copy-only backup, compress backups, and encrypt backups.

This lesson explores these options.

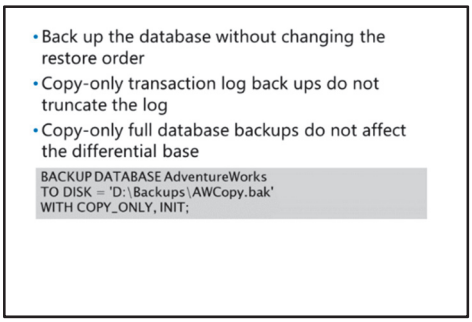
Lesson Objectives

After completing this lesson, you will be able to:

- Perform a copy-only backup.
- Use backup compression.
- Use backup encryption.

Copy-Only Backups

A copy-only SQL Server backup is independent of the sequence of conventional SQL Server backups. Usually, taking a backup changes the database and affects how later backups are restored. However, you may need to take a backup for a special purpose without affecting the overall backup and restore procedures for the database.



Create a copy-only independent database backup:

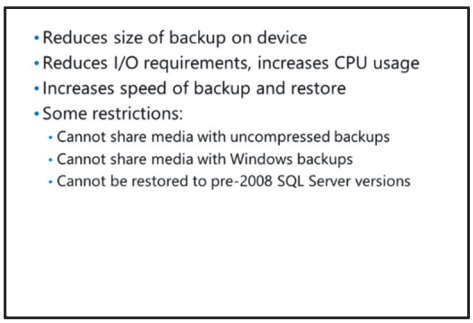
Copy-only Backup

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW_Copy.bak'
WITH COPY_ONLY, INIT;
```

You can make copy-only backups of either the database or the transaction logs. Restoring a copy-only full backup is the same as restoring any full backup.

Compressing Backups

Backup files can quickly become very large, so SQL Server gives you the ability to compress them. You can set the default backup compression behavior and also override this setting for individual backups. The following restrictions apply to compressed backups:



- Compressed and uncompressed backups cannot coexist in a media set.
- Windows-based backups cannot share a media set with compressed SQL Server backups.

- Backup compression is only available in the Enterprise and Standard Editions of SQL Server.

You can use the property pages for the server to view and configure the default backup compression setting.

To compress a backup, you can use the WITH COMPRESSION option of the BACKUP statement:

Compressing a Backup

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW_Comp.bak'
WITH COMPRESSION;
```

If your default setting is to compress backups and you want to override this, use the NO_COMPRESSION option.

The compression level that can be achieved depends upon how compressible the data is in the database. Some data compresses well, other data does not. A reduction in I/O and backup size of 30 to 50 percent is not uncommon in typical business systems.

 **Note:** Backup compression can be used on a database that is encrypted by using Transparent Database Encryption (TDE), though the compression rate will be minimal.

Impact of Compressed Backups on Performance

Because a compressed backup is smaller than an uncompressed backup of the same amount of data, compressing a backup typically reduces the amount of device I/O required and significantly decreases the duration of backups.

However, any form of compression tends to increase CPU usage. The additional CPU resources that are consumed by the compression process may adversely impact concurrent operations on systems that are CPU bound. Most current SQL Server systems are I/O bound, rather than CPU bound, so the benefit of reducing I/O usually outweighs the increase in CPU requirements by a significant factor.

Impact of Compression on Recovery Time

Although a reduction in the time taken to perform backups can be beneficial, backups are usually performed while the system is being used and should not impact availability. However, compression benefits not only the backup process but also the restore process, and can significantly improve the ability to meet RTO requirements.

Demonstration: Using Backup Compression

In this demonstration, you will see how to use backup compression.

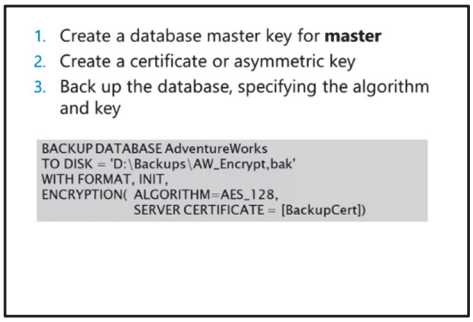
Demonstration Steps

Use Backup Compression

1. In SQL Server Management Studio, in Object Explorer, under **Databases**, right-click **AdventureWorks**, point to **Tasks**, and click **Back Up**.
2. In the **Back Up Database - AdventureWorks** dialog box, ensure that **Backup type** is set to **Full**.
3. In the **Destination** section, select the existing file path, click **Remove**, and then click **Add**.
4. In the **Select Backup Destination** dialog box, in the **File name** box, type **D:\Demofiles\Mod06\Demo\AW_Comp.bak**, and then click **OK**.

5. In the **Back Up Database - AdventureWorks** dialog box, on the **Media Options** page, note that the default option is to append to an existing media set. In this case, there is no existing media set, so a new one will be created—there are no existing backup sets to overwrite.
6. In the **Back Up Database - AdventureWorks** dialog box, on the **Backup Options** page, change the **Name** to **AdventureWorks-Compressed Backup**.
7. In the **Set backup compression** list, click **Compress backup**.
8. In the **Back Up Database - AdventureWorks** dialog box, in the **Script** drop-down list, click **Script Action to New Query Window**, and then click **OK**.
9. When the backup has completed successfully, click **OK**.
10. In the query pane, view the Transact-SQL BACKUP statement that was used to back up the database, noting that the COMPRESSION option was specified.
11. View the **D:\Demofiles\Mod06\Demo** folder and note the size of the **AW_Comp.bak** file. This should be significantly smaller than the **AW.bak** file was after the full database backup in the previous demonstration.
12. Keep SQL Server Management Studio open for the next demonstration.

Encrypting Backups



Backups are a fundamental requirement for protecting an organization's data against hardware failure or natural disaster. However, the data in the backup may be sensitive, and you must ensure that the backup media is secured against unauthorized access to the data it contains. In most organizations, you can accomplish this goal by storing backup media in secured file system locations. However, it is common for organizations to use an offsite storage solution for backups to protect against loss of data in the event of a disaster that affects the entire site (for example, a flood or fire). In this kind of scenario, or when the data in the backup requires additional security for compliance reasons, you can encrypt backups so that they can only be restored on a SQL Server instance that contains the correct encryption key. Backup encryption in SQL Server is based on standard encryption algorithms, including AES 128, AES 192, AES 256, and Triple DES. To encrypt a backup, you must specify the algorithm you want to use and a certificate or asymmetric key that can be used to encrypt the data.

To use backup encryption:

1. Create a database master key in the master database. This is a symmetric key that is used to protect all other encryption keys and certificates in the database.
2. Create a certificate or asymmetric key with which to encrypt the backup. You can create a certificate or asymmetric key in a SQL Server database engine instance by using the CREATE CERTIFICATE or CREATE ASYMMETRIC KEY statement. Note that asymmetric keys must reside in an extended key management (EKM) provider.
3. Perform the backup using the ENCRYPTION option (or select **Encryption** in the **Backup Database** dialog box), and specifying the algorithm and certificate or asymmetric key to be used. When using the **Backup Database** dialog box, you must select the option to back up to a new media set.

You should back up the database master key and encryption keys to a secure location (separate from the backup media location) so that you can restore the database to a different SQL Server instance in the event of a total server failure.

The following example code backs up the **AdventureWorks** database using the AES 128 encryption algorithm and a certificate named **BackupCert**:

Using Backup Encryption

```
BACKUP DATABASE AdventureWorks
TO DISK = 'D:\Backups\AW_Encrypt,bak'
WITH FORMAT, INIT,
ENCRYPTION(   ALGORITHM=AES_128,
              SERVER CERTIFICATE = [BackupCert])
```

Demonstration: Using Backup Encryption

In this demonstration, you will see how to:

- Create a database master key.
- Create a certificate.
- Encrypt a database backup.

Demonstration Steps

Create a Database Master Key

1. Using File Explorer, create a folder called **D:\Backups**.
2. In SQL Server Management Studio, open the **EncyptionKeys.sql** script file in the **D:\Demofiles\Mod06\Setupfiles** folder.
3. Select the code under the comment **Create a database master key** and click **Execute**.
4. Select the code under the comment **Back up the database master key** and click **Execute**.

Create a Certificate

1. Select the code under the comment **Create a certificate** and click **Execute**.
2. Select the code under the comment **Back up the certificate and its private key** and click **Execute**.

Encrypt a Database Backup

1. In Object Explorer, under **Databases**, right-click **AdventureWorks**, point to **Tasks**, and then click **Back Up**.
2. In the **Back Up Database - AdventureWorks** dialog box, ensure that **Backup type** is set to **Full**.
3. In the **Destination** section, select the existing file path, click **Remove**, and then click **Add**.
4. In the **Select Backup Destination** dialog box, in the **File name** box, type **D:\Backups\AW_Encrypt.bak**, and then click **OK**.
5. In the **Back Up Database - AdventureWorks** dialog box, on the **Media Options** page, click **Back up to a new media set, and erase all existing backup sets**.
6. In the **New media set name** box, type **Encrypted Backup**.
7. In the **Back Up Database - AdventureWorks** dialog box, on the **Backup Options** page, change the **Name** to **AdventureWorks-Encrypted Backup**.

8. In the **Set backup compression** list, click **Compress backup**.
9. In the **Encryption** section, select the **Encrypt backup** check box, ensure that the **AES 256** algorithm is selected, and select the **AdventureWorks** certificate you created previously.
10. In the **Back Up Database - AdventureWorks** dialog box, in the **Script** drop-down list, click **Script Action to New Query Window**, and then click **OK**.
11. When the backup has completed successfully, click **OK**.
12. In the query pane, view the Transact-SQL BACKUP statement that was used to back up the database, noting that the ENCRYPTION option was specified.
13. Close SSMS without saving any changes.

Lab: Backing Up Databases

Scenario

As a database administrator for Adventure Works Cycles, you are responsible for the **AdventureWorks** regional, national, and data archive databases. You must implement a backup solution for these databases, based on the backup requirements that have been provided.

Objectives

After completing this lab, you will be able to:

- Implement a backup strategy based on full database backups.
- Implement a backup strategy based on full, differential, and transaction log backups.
- Implement a backup strategy based on filegroup and partial backups.

Estimated Time: 90 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Backing Up Databases

Scenario

The backup strategy for the regional **AdventureWorks** database is based on daily full database backups. Daily backups are sufficient at regional level, as stock levels and sales for the day can be quickly recovered from the till records. You must perform tests of these backup operations and verify the backups.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Set the Recovery Model
3. Perform a Full Database Backup
4. Modify Data in the Database
5. Perform Another Full Database Backup
6. View the Backup and Restore Events Report

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab06** folder as Administrator.
3. Create the **D:\Backups** folder.

▶ Task 2: Set the Recovery Model

1. Based on the proposed backup strategy for the **AdventureWorks** database, determine the appropriate recovery model for the database.
2. Use SQL Server Management Studio to check the current recovery model of the database, and change it if necessary.

▶ Task 3: Perform a Full Database Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AdventureWorks.bak**.
   - Use a full backup.
   - Create a new media set with the name **AdventureWorks Backup**.
   - Name the backup set **AdventureWorks-Full Database Backup**.
   - Compress the backup.
2. Verify that the backup file has been created, and note its size.

▶ Task 4: Modify Data in the Database

- Update the **Employee** table in the **AdventureWorks** database using the following Transact-SQL code:

```
UPDATE HumanResources.Employee
SET VacationHours = VacationHours + 10 WHERE SickLeaveHours < 30;
```

▶ Task 5: Perform Another Full Database Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AdventureWorks.bak**.
   - Use a full backup.
   - Back up the database to the existing media set, and append the backup to the existing backup sets.
   - Name the backup set **AdventureWorks-Full Database Backup 2**.
   - Compress the backup.
2. Verify that the size of the backup file has increased.

▶ Task 6: View the Backup and Restore Events Report

1. In SQL Server Management Studio, view the **Backup and Restore Events** report for the **AdventureWorks** database.
2. Verify that the report shows the two backups you have created, **AdventureWorks-Full Database Backup** and **AdventureWorks-Full Database Backup 2**.

Results: At the end of this exercise, you will have backed up the **AdventureWorks** database to D:\Backups\AdventureWorks.bak using the simple recovery model.

Exercise 2: Performing Database, Differential and Transaction Log Backups

Scenario

The backup strategy for the national **AdventureWorks** database uses a combination of full, differential, and transaction log backups.

The main tasks for this exercise are as follows:

1. Set the Recovery Model
2. Perform a Full Database Backup
3. Modify Data in the Database
4. Perform a Transaction Log Backup
5. Modify Data in the Database
6. Perform a Differential Backup
7. Modify Data in the Database
8. Perform Another Transaction Log Backup
9. Verify Backup Media

▶ Task 1: Set the Recovery Model

1. Based on the proposed backup strategy for the **AdventureWorks** database, determine the appropriate recovery model for the database.
2. Use SQL Server Management Studio to check the current recovery model of the database, and change it if necessary.

▶ Task 2: Perform a Full Database Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AWNational.bak**.
   - Use a full backup.
   - Create a new media set with the name **AdventureWorks Backup**.
   - Name the backup set **AdventureWorks-Full Database Backup**.
   - Compress the backup.
2. Verify that the backup file has been created, and note its size.

▶ Task 3: Modify Data in the Database

- Update the **Employee** table in the **AdventureWorks** database using the following Transact-SQL code:

```
UPDATE HumanResources.Employee
SET VacationHours = VacationHours + 10 WHERE SickLeaveHours < 30;
```

▶ Task 4: Perform a Transaction Log Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AWNational.bak**.
   - Use a transaction log backup.
   - Back up the log to the existing media set, and append the backup to the existing backup sets.
   - Name the backup set **AdventureWorks-Transaction Log Backup**.
   - Compress the backup.
2. Verify that the size of the backup file has increased.

▶ Task 5: Modify Data in the Database

- Update the **Employee** table in the **AdventureWorks** database using the following Transact-SQL code:

```
UPDATE HumanResources.Employee
SET VacationHours = VacationHours + 10 WHERE SickLeaveHours < 30;
```

▶ Task 6: Perform a Differential Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AWNational.bak**.
   - Use a differential backup.
   - Back up the log to the existing media set, and append the backup to the existing backup sets.
   - Name the backup set **AdventureWorks-Differential Backup**.
   - Compress the backup.
2. Verify that the size of the backup file has increased.

▶ Task 7: Modify Data in the Database

- Update the **Employee** table in the **AdventureWorks** database using the following Transact-SQL code:

```
UPDATE HumanResources.Employee
SET VacationHours = VacationHours + 10 WHERE SickLeaveHours < 30;
```

▶ Task 8: Perform Another Transaction Log Backup

1. Back up the **AdventureWorks** database to **D:\Backups\AWNational.bak**.
   - Use a transaction log backup.
   - Back up the log to the existing media set, and append the backup to the existing backup sets.
   - Name the backup set **AdventureWorks-Transaction Log Backup 2**.
   - Compress the backup.
2. Verify that the size of the backup file has increased.

▶ Task 9: Verify Backup Media

1. Use the following query to verify that the backups you performed in this exercise are all on the backup device:

```
RESTORE HEADERONLY
FROM DISK = 'D:\Backups\AWNational.bak';
GO
```

2. Use the following query to identify the database files that are included in the backups:

```
RESTORE FILELISTONLY
FROM DISK = 'D:\Backups\AWNational.bak';
GO
```

3. Use the following query to verify that the backups are valid:

```
RESTORE VERIFYONLY
FROM DISK = 'D:\Backups\AWNational.bak';
GO
```

Results: At the end of this exercise, you will have backed up the national database to D:\Backups\AWNational.bak.

Exercise 3: Performing a Partial Backup

Scenario

The **AdventureWorks** data archive database is too large for a conventional backup strategy so a partial backup strategy is to be put in place.

The main tasks for this exercise are as follows:

1. Create Read-Only Filegroup
2. Back Up the Read-Only Filegroup
3. Perform a Partial Backup
4. Modify Data in the Database
5. Perform a Differential Partial Backup
6. Verify Backup Media

▶ Task 1: Create Read-Only Filegroup

1. In SQL Server Management Studio, open the **CreateReadOnly.sql** script file from the **D:\Labfiles\Lab06\Starter\Setupfiles** folder, and then click **Execute**.
2. This script creates the required read-only components you need for this Lab, and then close the query pane without saving the file.

▶ Task 2: Back Up the Read-Only Filegroup

1. Use the following query to back up the read-only filegroup:

```
BACKUP DATABASE AdventureWorks FILEGROUP = 'ReadOnlyFG' TO DISK =
'D:\Labfiles\Lab06\Starter\AWReadOnly.bak' WITH FORMAT, INIT, NAME = 'AdventureWorks
Read Only FG Backup', COMPRESSION;
```

2. Verify that the backup file **AWReadOnly.bak** has been created in the **D:\Labfiles\Lab06\Starter** folder.

▶ Task 3: Perform a Partial Backup

1. Use the following query to perform a partial backup of all read-write filegroups:

```
BACKUP DATABASE AdventureWorks READ_WRITE_FILEGROUPS TO DISK =
'D:\Labfiles\Lab06\Starter\AWPartial.bak' WITH FORMAT, INIT, NAME = 'AdventureWorks
Partial Backup', COMPRESSION;
```

2. Verify that the backup file **AWPartial.bak** has been created in the **D:\Labfiles\Lab06\Starter** folder.

▶ Task 4: Modify Data in the Database

- Update the **Employee** table in the **AdventureWorks** database using the following Transact-SQL code:

```
UPDATE HumanResources.Employee
SET VacationHours = VacationHours + 10 WHERE SickLeaveHours < 30;
```

▶ Task 5: Perform a Differential Partial Backup

1. Use the following query to perform a differential partial backup of the **AdventureWorks** database:

```
BACKUP DATABASE AdventureWorks READ_WRITE_FILEGROUPS TO DISK = '
D:\Labfiles\Lab06\Starter\AWPartialDifferential.bak' WITH DIFFERENTIAL, NOFORMAT,
NOINIT, NAME = 'AdventureWorks-Active Data Diff', COMPRESSION;
```

2. Verify that the backup file **AWPartialDifferential.bak** has been created in the **D:\Labfiles\Lab06\Starter** folder.

▶ Task 6: Verify Backup Media

1. Use the following query to view the backups on **AWReadOnly.bak**, and scroll to the right to view the **BackupTypeDescription** column:

```
RESTORE HEADERONLY FROM DISK = 'D:\Labfiles\Lab06\Starter\AWReadOnly.bak';
GO
```

2. Use the following query to view the backups on **AWPartial.bak**, and scroll to the right to view the **BackupTypeDescription** column:

```
RESTORE HEADERONLY FROM DISK = 'D:\Labfiles\Lab06\Starter\AWPartial.bak';
GO
```

Results: At the end of this exercise, you will have backed up the read-only filegroup in the **AdventureWorks** database to **D:\Backups\AWReadOnly.bak**; and you will have backed up the writable filegroups in the **AdventureWorks** database to **D:\Backups\AWReadWrite.bak**.

Module Review and Takeaways

In this module, you have learned how to create a backup strategy that is aligned with organizational needs. You have also seen how the transaction logging capabilities of SQL Server can help you to achieve an appropriate outcome.

Best Practice:

- Plan your backup strategy carefully.
- Plan the backup strategy in conjunction with the business needs.
- Choose the appropriate database recovery model.
- Plan your transaction log size based on the transaction log backup frequency.
- Consider using differential backups to speed recovery.

Consider compressing backups to reduce storage requirements and backup time.

Review Question(s)

Question: What are the unique features of transaction log restores?

Question: When might a full database backup strategy be adequate?

Module 7

Restoring SQL Server Databases

Contents:

Module Overview

In the previous module, you learned how to create backups of Microsoft® SQL Server® databases. A backup strategy might involve many different types of backup, so it is essential that you can effectively restore them.

You will often be restoring a database in an urgent situation. You must, however, ensure that you have a clear plan of how to proceed and successfully recover the database to the required state. A good plan and understanding of the restore process can help avoid making the situation worse.

Some database restores are related to system failure. In these cases, you will want to return the system as close as possible to the state it was in before the failure. Some failures, though, are related to human error and you might wish to recover the system to a point before that error. The point-in-time recovery features of SQL Server can help you to achieve this.

Because they are typically much larger, user databases are more likely to be affected by system failures than system databases. However, system databases can be affected by failures, and special care should be taken when recovering them. In particular, you need to understand how to recover each system database because you cannot use the same process for all system databases.

In this module, you will see how to restore user and system databases and how to implement point-in-time recovery.

Objectives

After completing this module, you will be able to:

- Explain the restore process.
- Restore databases.
- Perform advanced restore operations.
- Perform a point-in-time recovery.

Lesson 1
Understanding the Restore Process

When you need to recover a database, you must have a good plan to avoid causing further damage. After you have completed the preliminary step of attempting to create a tail-log backup, it is most important to determine which database backups to restore—and their order.

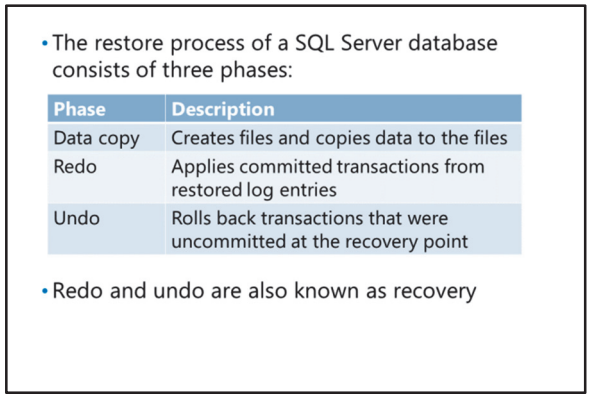
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the restore process.
- Describe the different types of restores.
- Decide on the backups to restore and in which order.

Phases of the Restore Process

Restoring a SQL Server database consists of three phases: *data copy*, *redo*, and *undo*. The combination of the redo and the undo phases is commonly referred to as the *recovery* of a database.



Data Copy

The data copy phase is typically the longest in a database restore. First, the data files from the database need to be retrieved from the backups. Before any data pages are restored, the restore process reads the header of the backup and SQL Server recreates the required data and log files. If instant file initialization (IFI) has not been enabled by granting rights to the SQL Server service account, the rewriting of the data files can take a substantial amount of time.

After the data and log files are recreated, the data files are restored from the full database backup. Data pages are retrieved from the backup in order and written to the data files. The log files need to be zeroed out before they can be used—if the log files are large, this process can also take a substantial time.

If a differential backup is also being restored, SQL Server overwrites the extents in the data files with those contained in the differential backup.

Redo Phase

At the start of the redo phase, SQL Server retrieves details from the transaction log. In the simple recovery model, these details are retrieved from either the full database backup or the differential backup. In the full or bulk-logged recovery model, these log file details are supplemented by the contents of any transaction log backups that were taken after the full and differential database backups.

In the redo phase, SQL Server rolls all changes that are contained within the transaction log details into the database pages, up to the recovery point. Typically, the recovery point is the latest time for which transactions exist in the log.

Undo Phase

The transaction log will likely include details of transactions that were not committed at the recovery point, which, typically, is the time of the failure. In the undo phase, SQL Server rolls back any of these uncommitted transactions.

Because the action of the undo phase involves rolling back uncommitted transactions and placing the database online, no more backups can be restored.

During the undo phase, the Enterprise edition of SQL Server will allow the database to come online so that users can begin to access it. This capability is referred to as the fast recovery feature. Queries that attempt to access data that is still being undone are blocked until the undo phase is complete. Potentially, this can cause transactions to time out, but does mean that users can access the database sooner.

In general, you cannot bring a database online until it has been recovered. The one exception to this is the fast recovery option, which allows users to access the database while the undo phase is continuing.

Recovery does not only occur during the execution of RESTORE commands. If a database is taken offline and then placed back into an ONLINE state, recovery of the database will also occur. The same recovery process takes place when SQL Server restarts.

Note: Other events that lead to database recovery include clustering or database mirroring failovers. Failover clustering and database mirroring are advanced topics that are beyond the scope of this course.

Types of Restores

The restore scenarios available for a database depend on its recovery model and the edition of SQL Server you are using.

- Complete database restores:
  - Simple recovery model
  - Full recovery model
- System database restore
- Advanced restores:
  - File or filegroup restore
  - Piecemeal restore
  - Encrypted backup restore
  - Page restore

Complete Database Restore in Simple Recovery Model

The most basic restore strategy for SQL Server databases is to restore and recover a full database backup. If available, you can restore the latest differential backup after the restore of the full database backup but before the recovery process for the database.

In most scenarios involving the simple recovery model, no differential backups are performed. In these cases, you only restore the last full database backup, after which the recovery phase returns the database to the state it was in at the time just before the full database backup was completed.

Complete Database Restore in Full Recovery Model

The most common restore strategy requires full or bulk-logged recovery model and involves restoring full, differential (if present), and log backups. While the aim of the restore is to recover the database to the latest point in time possible, options exist to restore the database to an earlier time. These options will be discussed later in this module.

System Database Restore

You can restore system databases but this requires special processes to prevent further issues from occurring. For example, if a master database is left in an inconsistent state, SQL Server will refuse to start until the master database is recovered. The recovery of system databases will be discussed later in this module.

Filegroup or File Restore

SQL Server includes the functionality to restore filegroups or individual files; therefore, if specific files in a database are corrupt or lost, you can potentially reduce the overall time to recover the database. The recovery of individual files is only supported for read-only files when operating in simple recovery model, but you can use it for read/write files when using the bulk-logged or full recovery models. The recovery of individual files uses a process that is similar to the complete database restore process. It is discussed later in this module.

Piecemeal Restore

A piecemeal restore is used to restore and recover the database in stages, based on filegroups, rather than restoring the entire database at a single time. The primary filegroup is the first that must be restored, usually along with the read/write filegroups for which you want to prioritize recovery. You can then restore read-only filegroups.

Page Restore

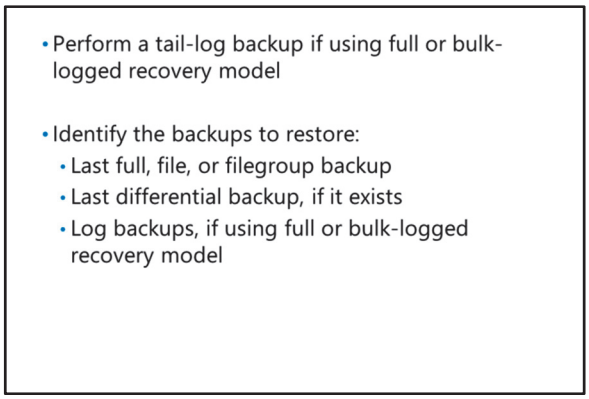
Another advanced option provides the ability to restore an individual data page. If an individual data page is corrupt, users will usually see either an 823 error or an 824 error when they execute a query that tries to access the page. You can try to recover the page using a page restore. If a user query tries to access the page after the restore starts, they will see error 829, which indicates the page is restoring. If the page restore is successful, user queries that access the page will again return results as expected. Page restores are supported under full and bulk-logged recovery models, but not under simple recovery model.

Online Restore

Online restore involves restoring data while the database is online. This is the default option for File, Page, and Piecemeal restores.

Preparations for Restoring Backups

Before restoring any database, it is important to attempt a tail-log backup, unless you are intending to replace the current state of the database. You can often perform a tail-log backup, even when damage has occurred to the data files of the database. The tail-log backup is critical when you need to restore the database to the latest possible point in time.



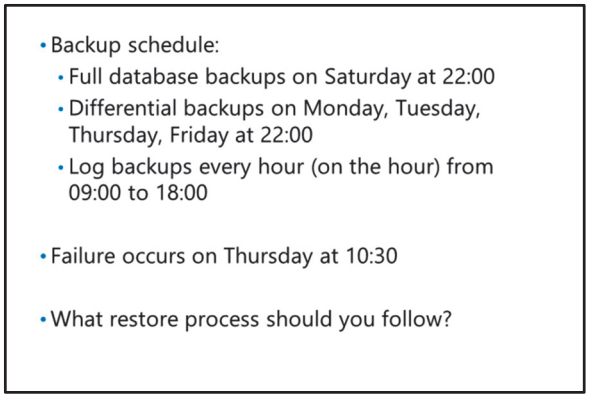
Identifying Backups to Restore

The recovery of any database depends upon restoring the correct backups in the correct order. The normal process for restoring a database is:

1. Restore the latest full database backup as a base to work from. If only individual files are damaged or missing, you may be able to restore just those files.
2. If there are differential backups, you only need to restore the latest differential backup.
3. If transaction log backups exist, you need to restore all transaction log backups since the last differential backup. You also need to include the tail-log backup created at the start of the restore process—if the tail-log backup was successful. (This step does not apply to databases using the simple recovery model.)

Discussion: Determining the Required Backups to Restore

The scenario on the slide describes the backup schedule for an organization and the timing of a failure. What restore process should you follow?



Lesson 2
Restoring Databases

Most restore operations involve restoring a full database backup, often followed by a differential backup, and a sequence of transaction log backups. In this lesson, you will learn how to restore these types of backup and recover a database.

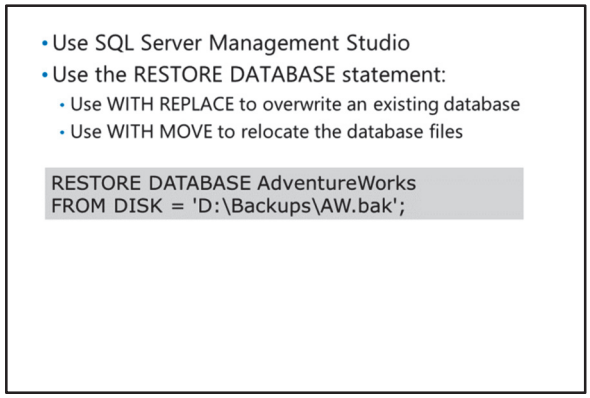
Lesson Objectives

After completing this lesson, you will be able to:

- Restore a full database backup.
- Restore a differential backup.
- Restore a transaction log backup.

Restoring a Full Database Backup

You can restore a database by using either SQL Server Management Studio (SSMS) or the RESTORE DATABASE statement in Transact-SQL.



Restoring a Database

The simplest recovery scenario is to restore a database from a single full database backup. If no subsequent differential or transaction log backups need to be applied, you can use the RECOVERY option to specify that SQL Server should complete the recovery process for the database and bring it online. If additional backups must be restored, you can prevent recovery from occurring by specifying the NORECOVERY option. If you do not specify either of these options, SQL Server uses RECOVERY as the default behavior.

In the following example, the **AdventureWorks** database is restored from the **AW.bak** backup media:

Restoring a Database from a Full Backup

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak';
```

Replacing an Existing Database

SQL Server will not allow you to restore a database backup over an existing database if you have not performed a tail-log backup on the database. If you attempt to do this using SSMS, SQL Server will issue a warning and will automatically attempt to create a tail-log backup for you. If you need to perform the restore operation and you do not have a tail-log backup, you must specify the WITH REPLACE option.

In the following code example, the existing **AdventureWorks** database is replaced with the database in the **AW.bak** backup media:

Replace Database

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak
WITH REPLACE';
```

Note: The WITH REPLACE option needs to be used with caution as it can lead to data loss.

Restoring Database Files to a Different Location

When you restore a database from another server, you might need to place the database files in different locations to those recorded in the backup from the original server. You might also need to do this if you are copying a database by using backup and restore. You can use the WITH MOVE option to specify new file locations.

In this example, the **AdventureWorks** database is being restored from another server. In addition to specifying the source location for the media set, new locations for each database file are also specified in the RESTORE statement. Note that the MOVE option requires the specification of the logical file name, rather than the original physical file path.

WITH MOVE

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH MOVE 'AdventureWorks_Data' TO 'Q:\Data\AdventureWorks.mdf',
          MOVE 'AdventureWorks_Log' TO 'U:\Logs\ AdventureWorks.ldf';
```

Restoring a Database in Standby Mode

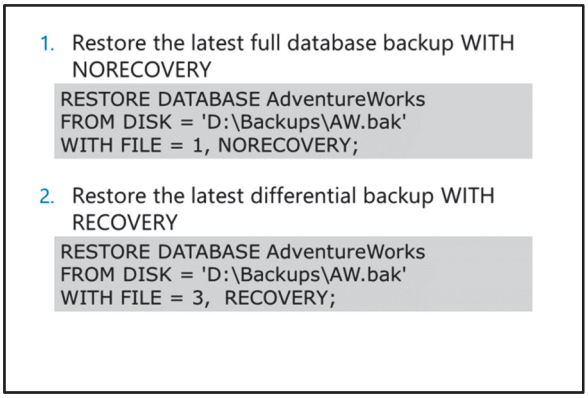
SQL Server provides the ability to view the contents of a database that has not been recovered—by using the option WITH STANDBY, instead of WITH NORECOVERY. After you restore a database by using the WITH STANDBY option, you can still apply further transaction log backups to the database. The STANDBY option is typically used to support log shipping scenarios, in which a secondary copy of a database is synchronized by reapplying the transactions in the transaction log of the primary database. You can also use the STANDBY option to enable inspection of data in a database that you do not want to bring online.

Restoring a Differential Backup

In a differential backup strategy, you must restore an initial full database backup, and then restore the latest differential backup.

Database Recovery When Restoring Multiple Backups

As discussed previously, the recovery process in SQL Server is critical to the maintenance of transactional integrity. This requires that all transactions that had committed before the failure are recorded in the database—all transactions that had not committed are rolled back.



The RESTORE command includes an option to specify WITH RECOVERY or WITH NORECOVERY. The WITH RECOVERY option is the default action and does not need to be specified. This ensures that a database is brought online immediately after being restored from a full database backup. However, when your backup strategy requires you to restore additional backups subsequent to the full database backup, it is important to choose the correct option for each RESTORE command. In most cases, this process is straightforward. All restores must be performed WITH NORECOVERY except the last restore, which must be WITH RECOVERY. Until the final backup is restored with the RECOVERY option, the database name will display as <*Database name*> *(Restoring…)* in SSMS.

There is no way to restore additional backups after a WITH RECOVERY restore has been processed. If you accidentally perform a backup using the WITH RECOVERY option, you must restart the entire restore sequence.

Restoring a Database from a Differential Backup

The command for restoring a differential backup is identical to that for restoring a full database backup. Differential backups are often appended to the same file as the full database backup (in which case, you need to stipulate the specific file from the media set that you want to restore).

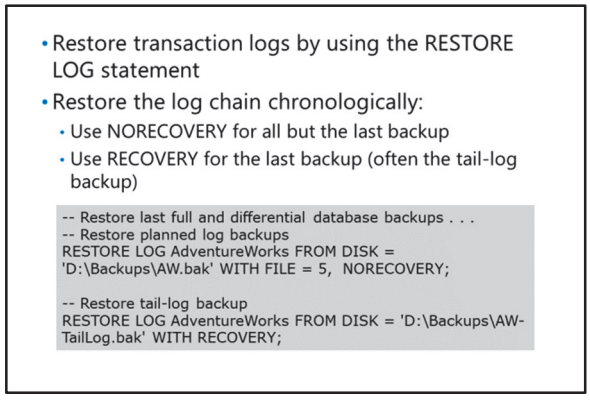
In this example, the **AdventureWorks** database is restored from the first file in the media set containing a full database backup. This media set is stored in the operating system file D:\Backups\AW.bak. The second file in the media set is the first differential backup, but the changes in this are also contained in the second differential backup, in the third file. Therefore, the second RESTORE statement only needs to restore the contents of the third file.

Restoring Full and Differential Backups

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 1, NORECOVERY;

RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 3,  RECOVERY;
```

If both the full and differential backup sets are on the same backup media, SSMS automatically selects the required backups in the **Restore Database** dialog box, and ensures that the appropriate recovery settings are applied.

Restoring Transaction Log Backups



You can restore the transaction logs for a database by using either SSMS or the RESTORE LOG statement in Transact-SQL. You should restore all log files, apart from the last log, using the WITH NORECOVERY option; then you should restore the last log file (which is often the tail-log backup) using the WITH RECOVERY option.

When using a backup strategy that involves transaction log backups, you must start by restoring the latest full database backup, followed by the latest differential backup if one exists. You must then restore all transaction logs created since the last full or differential backup in chronological order. Any break in the chain of

transaction logs will cause the restore process to fail, and require you to restart the recovery from the beginning.

In the following example, the **AdventureWorks** database has failed but the log file was accessible, so a tail-log backup has been stored in **AW-TailLog.bak**. To restore the database, the latest full backup (backup set **1** in **AW.bak**) is restored using the NORECOVERY option, followed by the latest differential backup (backup set **3** in **AW.bak**), again with the NORECOVERY option. All subsequent planned transaction log backups (backup sets **4** and **5** in **AW.bak**) are then restored in chronological order with the NORECOVERY option. Finally, the tail-log backup (the only backup set in **AW-TailLog.bak**) is restored with the RECOVERY option.

Using the RESTORE LOG Statement

```
-- Restore last full database backup
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 1, NORECOVERY;

-- Restore last differential backup
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 3,  NORECOVERY;

-- Restore planned log backups
RESTORE LOG AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 4,  NORECOVERY;

RESTORE LOG AdventureWorks
FROM DISK = 'D:\Backups\AW.bak'
WITH FILE = 5,  NORECOVERY;

-- Restore tail-log backup
RESTORE LOG AdventureWorks
FROM DISK = 'D:\Backups\AW-TailLog.bak'
WITH RECOVERY;
```

 **Note:** In the previous example, the log file was available after the database failed, so a tail-log backup could be taken. This means that the database can be recovered to the point of failure. Had the log file not been available, the last planned transaction log backup (backup set **5** in **AW.bak**) would have been restored using the RECOVERY option; all transactions since that backup would have been lost.

Demonstration: Restoring Databases

In this demonstration, you will see how to:

- Create a tail-log backup.
- Restore a database.

Demonstration Steps

Create a Tail-log Backup

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are running, and log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod07** folder, run **Setup.cmd** as Administrator.

3. In the **User Account Control** dialog box click **Yes**, and then wait until the script finishes.
4. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine using Windows authentication.
5. Click **New Query** and type the following Transact-SQL code to perform a tail-log backup:

```
BACKUP LOG AdventureWorks TO DISK = 'D:\Demofiles\Mod07\BackupDemo.bak'
WITH NO_TRUNCATE;
```

6. Click **Execute**, and view the resulting message to verify that the backup is successful.

Restore a Database

1. In Object Explorer, expand **Databases**, right-click **AdventureWorks**, point to **Tasks**, point to **Restore**, and then click **Database**.
2. In the **Restore Database - AdventureWorks** dialog box, note that the restore operation will restore both the full backup and the transaction log that you recently backed up, and then click **OK**.
3. In the **Microsoft SQL Server Management Studio** dialog box, note that the restore operation was successful, and then click **OK**.
4. In Object Explorer, verify that the **AdventureWorks** database is ready to use.
5. Close SQL Server Management Studio without saving any files.

Lesson 3
Advanced Restore Scenarios

The techniques discussed in the previous lesson cover most common restore scenarios. However, there are some more complex restore scenarios for which a DBA must be prepared.

This lesson discusses restore scenarios for file and filegroup backups, encrypted backups, individual data pages, and system databases.

Lesson Objectives

After completing this lesson, you will be able to:

- Restore a file or filegroup backup (including a piecemeal restore).
- Restore an encrypted backup.
- Restore individual data pages.
- Restore system databases.

Restoring File or Filegroup Backups

It can often be much quicker to restore a single file or filegroup than an entire database. You do not need to back up specific files or filegroups to restore them because SQL Server can extract the specific database files from a full or differential backup.

Restoring an individual file or filegroup:
1. Create a tail-log backup (if possible)
2. Restore each damaged file or filegroup
3. Restore any differential file backups
4. Restore transaction log backups in sequence
5. Recover the database

Performing a piecemeal restore:
1. Restore read/write filegroups with PARTIAL
2. Restore any differential or log backups and recover the database
3. Restore read-only filegroups

Restoring a File or Filegroup

Perform the following steps to restore an individual file or filegroup:

1. Create a tail-log backup of the active transaction log. (If you cannot do this because the log has been damaged, you must restore the whole database or restore to an earlier point in time.)
2. Restore each damaged file from the most recent file backup of that file.
3. Restore the most recent differential file backup, if any, for each restored file.
4. Restore transaction log backups in sequence, starting with the backup that covers the oldest of the restored files and ending with the tail-log backup created in step 1.
5. Recover the database.

To bring the database back to a consistent state, you must restore the transaction log backups that were created after the file backups. The transaction log backups can be rolled forward quickly, because only the changes that relate to the restored files or filegroups are applied. Undamaged files are not copied and then rolled forward. However, you still need to process the whole chain of log backups.

Performing a Piecemeal Restore

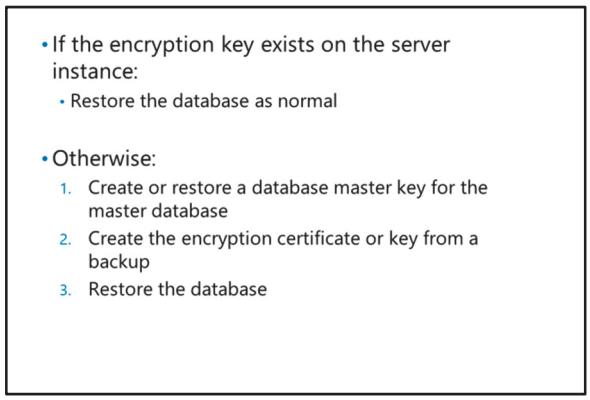
As discussed in the previous module, when a database is extremely large, you can use filegroups to store inactive data on read-only filegroups, and use a partial backup strategy in which each read-only filegroup is backed up once, and only read/write filegroups are included in subsequent backups—significantly reducing the time taken to perform a full or differential backup.

One of the advantages of including read-only filegroups in partial backup strategy is that it means you can perform a piecemeal restore. In a piecemeal restore, you can recover read/write filegroups, and make them available to users for querying before the recovery of read-only filegroups is complete. To perform a piecemeal restore:

1. Restore the latest partial full database backup, specifying the read/write filegroups to be restored and using the PARTIAL option to indicate that read-only filegroups will be restored separately.
2. Restore the latest partial differential backup, and log file backups if they exist. Use the RECOVERY option with the last RESTORE operation to recover the database. Data in read/write filegroups is now available.
3. Restore each read-only filegroup backup with the RECOVERY option to bring them online.

Restoring an Encrypted Backup

You can restore an encrypted backup to any SQL Server instance that hosts the certificate or key you used to encrypt the backup. This means that, if you need to restore an encrypted backup to the server from which you backed up the database, you can restore the backup using the same procedure as for a non-encrypted backup—providing the certificate or key still exists in the instance.



However, in many recovery scenarios, the database must be restored onto a different SQL Server instance; for example, because the original instance has failed irretrievably or because you want to move the database to a different server. In this case, you must use the following procedure to restore the encrypted database:

1. **Create a database master key for the master database**. This does not need to be the same database master key that was used in the original instance but, if you are recovering from a complete server failure, you can restore the original database master key from a backup.
2. **Create a certificate or key from a backup**. Use the CREATE CERTIFICATE or CREATE ASYMMETRIC KEY statement to create a certificate or key from the backup you created of the original key used to encrypt the database. The new certificate or key must have the same name as the original; if you used a certificate, you must restore both the public certificate and the private key.
3. **Restore the database**. Now that the encryption key is available on the SQL Server instance, you can restore the database as normal.

The following code sample shows how to restore an encrypted database backup on a new SQL Server instance:

Restoring an Encrypted Backup

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa55w.rd';

CREATE CERTIFICATE BackupCert
FROM FILE = 'K:\Backups\Backup.cer'
WITH PRIVATE KEY (
      DECRYPTION BY PASSWORD = 'CertPa55w.rd',
      FILE = 'K:\Backups\Backup.key');
GO

RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AW_Encrypt.bak'
```

Demonstration: Restoring an Encrypted Backup

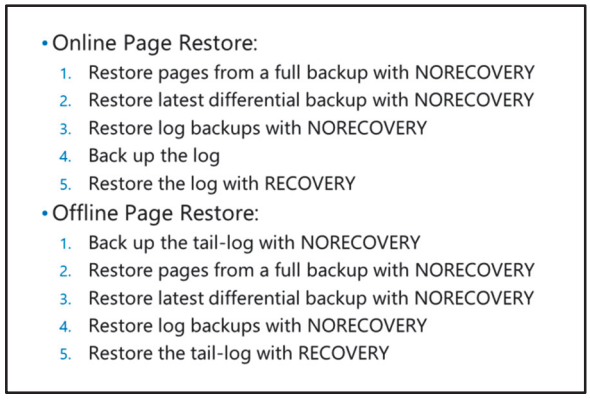
In this demonstration, you will see how to restore an encrypted backup.

Demonstration Steps

Restore an Encrypted Backup

1. Start SQL Server Management Studio and connect to the **MIA-SQL\SQL2** database engine using Windows authentication.
2. In Object Explorer, expand **Databases** and view the existing databases on this instance.
3. Open the **Restore Encrypted Backup.sql** script file in the **D:\Demofiles\Mod07** folder.
4. Select the code under the comment **Try to restore an encrypted backup** and click **Execute**. Note that this fails because the required certificate is not present.
5. Select the code under the comment **Create a database master key for master** and click **Execute**. This creates a database master key for the master database on **MIA-SQL\SQL2**.
6. Select the code under the comment **Import the backed up certificate** and click **Execute**. This creates a certificate from public and private key backups that were taken from the **MIA-SQL** instance.
7. Select the code under the comment **Restore the encrypted database** and click **Execute**. Note that this time the restore operation succeeds.
8. In Object Explorer, refresh the **Databases** folder and verify that the **AdventureWorks** database has been restored.
9. Close SQL Server Management Studio, without saving any changes.

Restoring Data Pages



In some scenarios, data pages in the database files can become corrupted. If this happens, you can restore individual pages to repair the database, which may be faster than restoring the entire database or the affected file. Page restore is only supported for a database using the full or bulk-logged recovery model.

When querying a table, the first indication of a corrupt page is usually the occurrence of error 823 or 824. To identify potentially corrupt pages, you can use the DBCC CHECKDB command and query the **suspect_pages** system table in the **msdb** database. This table provides the page ID of each affected page, along with details of the potential problem. With this information, you can restore damaged pages by using the RESTORE DATABASE Transact-SQL statement, or by using the **Restore Page** dialog box in SSMS.

To restore damaged pages while keeping the database online:

1. Restore one or more damaged pages from a full database backup.
2. Restore the latest differential backup if one exists.
3. Restore each subsequent transaction log backup with the NORECOVERY option.
4. Back up the transaction log.
5. Restore the transaction log with the RECOVERY option.

The final two steps of backing up and restoring the log are required to ensure that the final log sequence number (LSN) of the restored pages is set as the REDO target of the transaction log.

Online page restore is only supported in SQL Server Enterprise edition. You can perform an offline page restore by using the following procedure:

1. Back up the tail-log using the NORECOVERY option.
2. Restore one or more damaged pages from a full database backup.
3. Restore the latest differential backup if one exists.
4. Restore each subsequent transaction log backup with the NORECOVERY option.
5. Restore the tail-log with the RECOVERY option.

The following example code performs an online recovery of two pages:

Restoring a Page

```
-- Restore pages from the full backup
RESTORE DATABASE AdventureWorks PAGE='1:55, 1:207'
FROM DISK = 'D:\Backups\AdventureWorks.bak'
WITH FILE=1, NORECOVERY;

-- Apply the differential backup
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\Backups\AdventureWorks.bak'
WITH FILE=3, NORECOVERY;

-- Restore subsequent transaction log backups
RESTORE LOG AdventureWorks
FROM DISK = 'D:\Backups\AdventureWorks.bak'
WITH FILE=4, NORECOVERY;

-- Backup the log
BACKUP LOG AdventureWorks
TO DISK = 'D:\Backups\AW-Log.bak';

-- Restore the log to set the correct REDO LSN and recover
RESTORE LOG AdventureWorks
FROM DISK = 'D:\Backups\AW-Log.bak'
WITH RECOVERY;
```

Recovering System Databases

The recovery process for all system databases is not identical. Each system database has specific recovery requirements:

System database	Description
master	Backup required: yes Recovery model: simple Restore using single user mode
model	Backup required: yes Recovery model: user configurable Restore using –T3608 trace flag
msdb	Backup required: yes Recovery model: simple (default) Restore like any user database
tempdb/resource	No backups can be performed **tempdb** is created during instance startup Restore **resource** using file restore or setup

master

The **master** database holds all system-level configurations. SQL Server needs the **master** database before a SQL Server instance can run at all. SQL Server cannot start without the **master** database; therefore, if it is missing or corrupt, you cannot execute a standard RESTORE DATABASE command to restore it. Before starting to recover the **master** database, you must have access to a temporary **master** database so that the SQL Server instance will start. This temporary **master** database does not need to have the correct configuration—it will only be used to start up the instance to initiate the recovery process to restore the correct version of your **master** database. There are two ways that you can obtain a temporary master database:

- You can use the SQL Server setup program to rebuild the system databases, either from the location that you installed SQL Server from or by running the setup program found at Microsoft SQL Server\140\Setup Bootstrap\SQL 2017\setup.exe.

 **Note:** Rerunning the setup program will overwrite all your system databases, so you must ensure that they are regularly backed up and can be restored after you have restored the **master** database.

- You can use a file-level backup of the **master** database files to restore the **master** database. You must take this file-level backup when the **master** database is not in use—that is, when SQL Server is not running—or by using the VSS service.

 **Note:** Copying the **master** database from another instance is not supported. The VSS service is beyond the scope of this course.

When you have created a temporary version of the **master** database, you can use the following procedure to recover the correct **master** database:

1. Start the server instance in single-user mode by using the **–m** startup option.
2. Use a RESTORE DATABASE statement to restore a full database backup of the **master** database. It is recommended that you execute the RESTORE DATABASE statement by using the sqlcmd utility.

After restoring the **master** database, the instance of SQL Server will shut down and terminate your sqlcmd connection.

3. Remove the single-user startup parameter.
4. Restart SQL Server.

model

The **model** database is the template for all databases that are created on the instance of SQL Server. When the **model** database is corrupt, the instance of SQL Server cannot start. This means that a normal restore command cannot be used to recover the **model** database if it becomes corrupted. In the case of a corrupt **model** database, you must start the instance with the -T3608 trace flag as a command-line parameter. This trace flag only starts the **master** database. When SQL Server is running, you can restore the **model** database by using the normal RESTORE DATABASE command.

msdb

SQL Server Agent uses the **msdb** database for scheduling alerts and jobs, and for recording details of operators. The **msdb** database also contains history tables, such as those that record details of backup and restore operations. If the **msdb** database becomes corrupt, SQL Server Agent will not start. You can restore the **msdb** database by using the RESTORE DATABASE statement as you would a user database—the SQL Server Agent service can then be restarted.

resource

The **resource** database is read-only and contains copies of all system objects that ship with SQL Server. This is a hidden database and you cannot perform backup operations on it. It can, however, be corrupted by failures in areas such as I/O subsystems or memory. If the **resource** database is corrupt, it can be restored by a file-level restore in Windows or by running the setup program for SQL Server.

tempdb

The **tempdb** database is a workspace for holding temporary or intermediate result sets. This database is recreated every time an instance of SQL Server starts, so there is no need to back up or restore it. When the server instance is shut down, any data in **tempdb** is permanently deleted.

Lesson 4
Point-in-time Recovery

In the previous lesson, you learned how to recover a database to the latest point in time possible. However, you may need to recover the database to an earlier point in time. You have also learned that you can stop the restore process after any of the backups are restored and initiate the recovery of the database. While stopping the restore process after restoring an entire backup file provides a coarse level of control over the recovery point, SQL Server provides additional options that allow for more fine-grained control.

In this lesson, you will learn about how point-in-time recovery works and how to use the options that it provides.

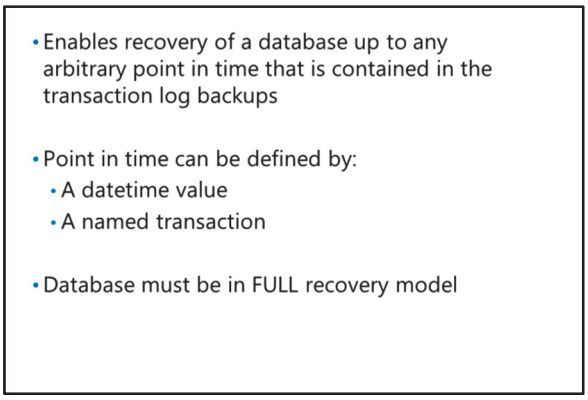
Lesson Objectives

After completing this lesson, you will be able to:

- Describe point-in-time recovery.
- Use the STOPAT restore option.
- Use the STOPATMARK restore option.
- Perform a point-in-time recovery in SSMS.

Overview of Point-in-time Recovery

You will mostly want to recover a database to the most recent point in time; however, sometimes you may want to restore it to an earlier stage. You can use SQL Server to restore a database to a specified point in time and then recover it. You can either restore it to an exact time by using a datetime value, or you can recover to a named transaction in the transaction log.



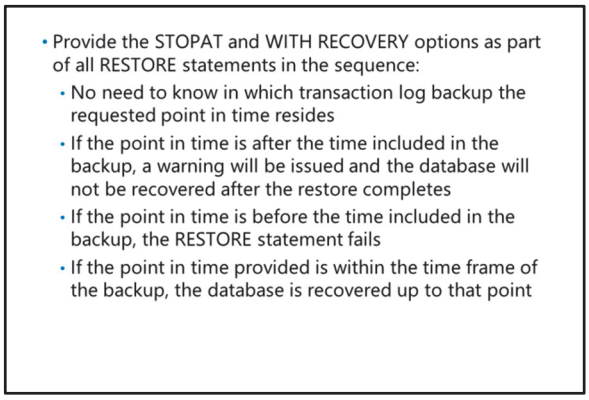
For either of these options to work, the database needs to use the full recovery model. SQL Server can only stop at points in the transaction log chain when the database is in full recovery model. If a database changes from the full recovery model to the bulk-logged recovery model to process bulk transactions—and is then changed back to the full recovery model—the recovery point cannot be in the time that the database was using the bulk-logged recovery model. If you attempt to specify a recovery point during which the database was using the bulk-logged recovery model, the restore will fail and an error will be returned.

Note: If a user error causes the inadvertent deletion of some data, you may not be aware of when the error actually occurred. Therefore, you will not know which log file contains the deletion and the point at which to recover the database. You can use the WITH STANDBY option on each log file restore and inspect the state of the database after each restore operation, to determine when the error occurred and when to recover the database.

STOPAT Option

You use the STOPAT option to specify a recovery point that is based on a datetime value. You might not know in advance which transaction log backup file contains transactions from the time where the recovery needs to occur. Therefore, you can use the syntax of the RESTORE LOG command to specify the RECOVERY option for each log restore command in the sequence.

In this example, the RECOVERY option is specified for each transaction log restore, but the actual recovery process will not take place until the time specified in the STOPAT option is located in the transaction log.



Using the STOPAT Option

```
RESTORE DATABASE database_name FROM full_backup
WITH NORECOVERY;

RESTORE DATABASE database_name FROM differential_backup
WITH NORECOVERY;

RESTORE LOG database_name FROM first_log_backup
WITH STOPAT = time, RECOVERY;

… (additional log backups could be restored here)

RESTORE LOG database_name FROM final_log_backup
WITH STOPAT = time, RECOVERY;
```

The behavior of the STOPAT and RECOVERY options is as follows:

- If the specified time is earlier than the first time in the transaction log backup, the restore command fails and returns an error.
- If the specified time is contained within the period covered by the transaction log backup, the restore command recovers the database at that time.
- If the specified time is later than the last time contained in the transaction log backup, the restore command restores the logs, sends a warning message, and the database is not recovered, so that additional transaction log backups can be applied.

This behavior ensures that the database is recovered up to the requested point, even when STOPAT and RECOVERY are both specified with every restore.

STOPATMARK Option

If you require more precise control over the recovery point, you can use the STOPATMARK option of the RESTORE Transact-SQL statement.

If you know in advance that you might need to recover to the point of a specific operation, you can place a mark in the transaction log to record the precise location. This example starts a transaction with the name and transaction mark of *UpdPrc* and a description of *Start of nightly update process*:

- Transactions marked using:
  - BEGIN TRAN <name> WITH MARK <description>
- Restore has two related options:
  - STOPATMARK rolls forward to the mark and includes the marked transaction in the roll forward
  - STOPBEFOREMARK rolls forward to the mark and excludes the marked transaction from the roll forward
- If the mark is not present in the transaction log backup, the backup is restored, but the database is not recovered

Marking a Transaction

```
BEGIN TRAN UpdPrc WITH MARK 'Start of nightly update process';
```

If you do not know the name of a transaction that was marked, you can query the **dbo.logmarkhistory** table in the **msdb** database.

The STOPATMARK option is similar to the STOPAT option for the RESTORE command. SQL Server will stop at the named transaction mark and include the named transaction in the redo phase. If you wish to exclude the transaction (that is, restore everything up to the beginning of the named transaction), you can use the STOPBEFOREMARK option instead. If the transaction mark is not found in the transaction log backup that is being restored, the restore completes and the database is not recovered, so that other transaction log backups can be restored.

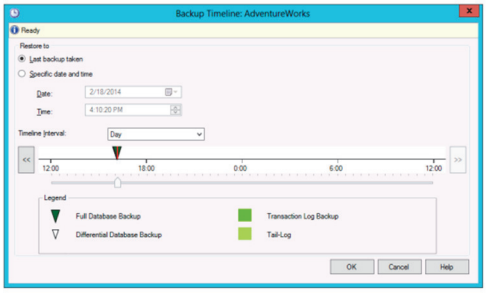
The STOPATMARK feature is mainly used when you want to restore an entire set of databases to a mutually consistent state, at some earlier point in time. If you need to perform a backup of multiple databases, so that they can all be recovered to a consistent point, consider marking all the transaction logs before commencing the backups.

Note: You cannot use the stop at mark functionality in SSMS; it is only available by using the Transact-SQL statement.

Performing a Point-in-time Recovery by Using SQL Server Management Studio

SQL Server Management Studio (SSMS) provides a GUI that makes it easy to restore a database to a specific point in time.

To perform a point-in-time recovery by using SSMS, follow the usual steps to open the **Restore Database** dialog box. On the **General** page, click **Timeline** to open the **Backup Timeline** dialog box, where you can configure a specific date and time where the restore should stop.



Demonstration: Performing a Point-in-time Recovery

In this demonstration you will see how to perform a point-in-time recovery.

Demonstration Steps

Perform a Point-in-time Recovery

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, open the **Point-in-Time Restore.sql** script file in the **D:\Demofiles\Mod07** folder.
3. Select and execute the code under the comment **Create a database and back it up**. This creates a database with a single table, and performs a full backup.
4. Select and execute the code under the comment **Enter some data**. This inserts a record into the **Customers** table.
5. Select and execute the code under the comment **Get the current time**. This displays the current date and time. Make a note of the current time.
6. Wait until a minute has passed, and then select and execute the code under the comment **Get the current time** again to verify that it is now at least a minute since you noted the time.
7. Select and execute the code under the comment **Enter some more data**. This inserts a second record into the **Customers** table.
8. Select and execute the code under the comment **Backup the transaction log**. This performs a transaction log backup of the database.
9. Close the query window.
10. In Object Explorer, expand **Databases** and verify that **BackupDemo** is listed (if not, right-click the **Databases** folder, and click **Refresh**).
11. Right-click the **BackupDemo** database, point to **Tasks**, point to **Restore**, and then click **Database**.
12. In the **Restore Database - BackupDemo** dialog box, click **Timeline**.
13. In the **Backup Timeline: BackupDemo** dialog box, select **Specific date and time** and set the **Time** value to the time you noted earlier (before any data was inserted), and then click **OK**.
14. In the **Restore Database - BackupDemo** dialog box, click **OK**.
15. When notified that the database has been restored successfully, click **OK**.
16. In Object Explorer, expand the **BackupDemo** database, expand the **Tables** folder, right-click **dbo.Customers**, and then click **Select Top 1000 Rows**. When the results are displayed, verify that the database was restored to the point in time before any data was inserted.
17. Close SQL Server Management Studio without saving any files.

Lab: Restoring SQL Server Databases

Scenario

You are a DBA with responsibility for managing the **HumanResources**, **InternetSales**, and **AWDataWarehouse** databases. You have backed up the databases according to their individual backup strategies, and you must now recover them in the event of a failure.

Objectives

After completing this lab, you will be able to:

- Restore a database from a full backup.
- Restore a database from full, differential, and transaction log backups.
- Perform a piecemeal restore of a large database.

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Restoring a Database Backup

Scenario

The **HumanResources** database has failed to come online; you must determine the problem and recover it to its last backed-up state.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Determine the Cause of the Failure
3. Restore the HumanResources Database

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab07\Starter** folder as Administrator.

▶ Task 2: Determine the Cause of the Failure

1. Use SQL Server Management Studio to view the status of the **HumanResources** database on the **MIA-SQL** instance of SQL Server.
2. Use the following Transact-SQL query to try to bring the database online:

```
ALTER DATABASE HumanResources SET ONLINE;
```

3. Review the error message, and then check the contents of the **D:\Labfiles\Lab07\Starter\Setupfiles** folder to determine if the **HumanResources.mdf** file is present. If not, the database cannot be brought online because the primary data file is lost.

▶ Task 3: Restore the HumanResources Database

1. Restore the **HumanResources** database from the most recent full backup.
2. Verify that the database has been restored.

Results: After this exercise, you should have restored the **HumanResources** database.

Exercise 2: Restoring Database, Differential, and Transaction Log Backups

Scenario

The **HumanResources** database has failed to come online—you must determine the problem and recover it to the most recent transaction possible.

The main tasks for this exercise are as follows:

1. Determine the Cause of the Failure
2. Perform a Tail-log Backup
3. Restore the InternetSales Database to Another Instance

▶ Task 1: Determine the Cause of the Failure

1. Use SQL Server Management Studio to view the status of the **InternetSales** database on the **MIA-SQL** instance of SQL Server.
2. Use the following Transact-SQL query to try to bring the database online:

```
ALTER DATABASE InternetSales SET ONLINE;
```

3. Review the error message, and then check the contents of the **D:\Labfiles\Lab07\Starter\Setupfiles** folder to verify that the **InternetSales.mdf** file is present. This file has become corrupt, and has rendered the database unusable.

▶ Task 2: Perform a Tail-log Backup

1. Verify that the **InternetSales.ldf** transaction log file is present in the **D:\Labfiles\Lab07\Starter\Setupfiles** folder.
2. Use the following Transact-SQL code to back up the tail of the transaction log:

```
USE master;
BACKUP LOG InternetSales TO DISK = 'D:\Labfiles\Lab07\Backups\InternetSales.bak'
WITH NO_TRUNCATE;
```

▶ Task 3: Restore the InternetSales Database to Another Instance

1. Restore the **InternetSales** database to the SQL2 instance from the planned backups in **D:\Labfiles\Lab07\Backups\InternetSales.bak** and the tail-log backup.

1. Move the data and log files to **D:\Labfiles\Lab07\Backups**.
2. In this case, the backup history for the database has been lost, so you must specify the backup media sets for the existing planned backups, in addition to the tail-log backup you have just performed.

   The planned backups should be restored using the NORECOVERY option, and then the tail-log backup should be restored using the RECOVERY option.
3. Verify that the database has been restored.

Results: After this exercise, you should have restored the **InternetSales** database.

Exercise 3: Performing a Piecemeal Restore

Scenario

The **AWDataWarehouse** has been accidentally dropped, and you must recover it as quickly as possible.

The main tasks for this exercise are as follows:

1. Begin a Piecemeal Restore

2. Restore Read/Write Filegroups and Bring the Database Online

▶ Task 1: Begin a Piecemeal Restore

1. Verify that the **AWDataWarehouse** database does not exist on **MIA-SQL\SQL2** instance.
2. Use the following Transact-SQL code to initiate a piecemeal restore:

```
USE master;
RESTORE DATABASE AWDataWarehouse FILEGROUP='Current'
FROM DISK = 'D:\Labfiles\Lab07\Backups\AWDataWarehouse.bak'
WITH REPLACE, PARTIAL, FILE = 1, NORECOVERY;
```

 **Note:** This code restores the primary filegroup and the Current filegroup from a full database backup on the AWDataWarehouse.bak media set. The PARTIAL option indicates that only the primary and named read/write filegroups should be restored, and the NORECOVERY option leaves the database in a restoring state, ready for subsequent restore operations of the read/write filegroup data.

3. Refresh the **Databases** folder in Object Explorer to verify that the database is in a restoring state.

▶ Task 2: Restore Read/Write Filegroups and Bring the Database Online

1. Use the following Transact-SQL code to restore a differential backup from the second backup set on the **AWDataWarehouse.bak** media set:

```
RESTORE DATABASE AWDataWarehouse
FROM DISK = 'D:\Labfiles\Lab07\Backups\AWDataWarehouse.bak'
WITH FILE = 2, RECOVERY;
```

2. Verify that the database is now shown as online in Object Explorer, and that you can query the **dbo.FactInternetSales** table.
3. Verify that you cannot query the **dbo.FactInternetSalesArchive** table, because it is stored in a filegroup that has not yet been brought online.

Results: After this exercise, you will have restored the **AWDataWarehouse** database.

Module Review and Takeaways

In this module, you learned how to restore databases from backups, including full database backups, differential database backups, and transaction log backups. You also learned how to restore individual filegroups and how to perform a piecemeal restore for a database that includes read/write and read-only filegroups.

When planning a database recovery solution, consider the following best practices:

Best Practice:

- Don't forget to back up the tail of the log before starting a restore sequence.
- If available, use differential restore to reduce the time taken by the restore process.
- Use file level restore to speed up restores when not all database files are corrupt.
- Perform regular database backups of **master**, **msdb** and **model** system databases.

Create a disaster recovery plan for your SQL Server and make sure you regularly test restoring databases.

Review Question(s)

Question: What are the three phases of the restore process?

Module 8

Automating SQL Server Management

Contents:

Module Overview

The tools provided by Microsoft® SQL Server® make administration easy when compared to some other database engines. However, even when tasks are easy to perform, it is common to have to repeat a task many times. Efficient database administrators learn to automate repetitive tasks. This can help to avoid situations where an administrator forgets to execute a task at the required time. Perhaps more importantly, the automation of tasks helps to ensure that they are performed consistently, each time they are executed.

This module describes how to use SQL Server Agent to automate jobs, how to configure security contexts for jobs, and how to implement multiserver jobs.

Objectives

After completing this module, you will be able to:

- Describe methods for automating SQL Server management.
- Create jobs, job step types, and schedules.
- Manage SQL Server Agent jobs.
- Configure master and target servers.

Lesson 1
Automating SQL Server Management

You can gain many benefits from the automation of SQL Server management. Most of the benefits center on the reliable, consistent execution of routine management tasks. SQL Server is a flexible platform that provides a number of ways to automate management, but the most important tool for this is the SQL Server Agent. All database administrators working with SQL Server must be familiar with the configuration and ongoing management of SQL Server Agent.

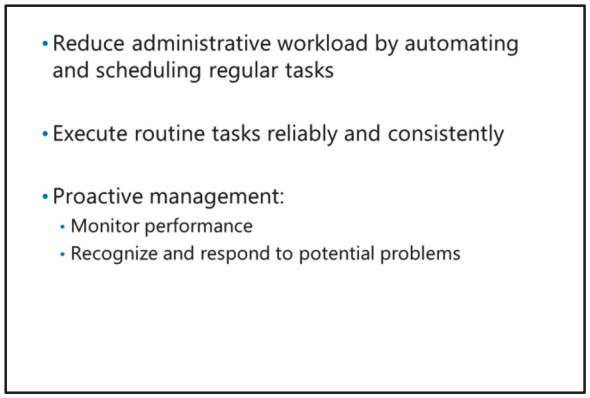
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the benefits of automating SQL Server management.
- Describe the available options for automating SQL Server management and the framework that SQL Server Agent provides.
- Effectively use SQL Server Agent.

Benefits of Automating SQL Server Management

All efficient database administrators automate their routine tasks. The benefits you can gain from the automation of SQL Server management include:



- **Reduced administrative load**. Unfortunately, some administrators who work with SQL Server, Windows®, and other tools, see their roles in terms of a constant stream of repetitive administrative tasks. For example, a Windows administrator at a university department might receive regular requests to create a large number of user accounts. The administrator might be happy to create each account one by one, using the standard tooling. A more efficient administrator learns to write a script to create users and execute that instead of performing the operation manually.

The same sort of situation occurs with routine tasks in SQL Server. While you can perform these tasks individually or manually, efficient database administrators do not do this. They automate all their routine and repetitive tasks. Automation removes the repetitive workload from the administrators so they can manage larger numbers of systems or perform higher-value tasks.

- **Reliable execution of routine tasks**. When you perform routine tasks manually, there is always a chance that you might overlook a vital task. For example, a database administrator could forget to perform database backups. By using automation, administrators can focus on exceptions that occur during the routine tasks, rather than on the execution of the tasks.
- **Consistent execution of routine tasks**. Another problem that can occur when you perform routine tasks manually is that you may not perform the tasks the same way each time. Imagine a situation where a database administrator archives some data from a set of production tables into a set of history tables every Monday morning. The new tables must have the same name as the originals with a suffix that includes the current date.

While the administrator might remember to perform this task every Monday morning, there is a possibility that one or more of the following errors could occur:

- Copy the wrong tables.
- Copy only some of the tables.
- Forget what the correct date is when creating the suffix.
- Format the date in the suffix incorrectly.
- Copy data into the wrong archive table.

Anyone who has been involved in the ongoing administration of systems will tell you that these and other problems will occur from time to time, even when the tasks are executed by experienced and reliable administrators. Automating routine tasks can assist greatly in making sure that they are performed consistently every time.

Proactive Management

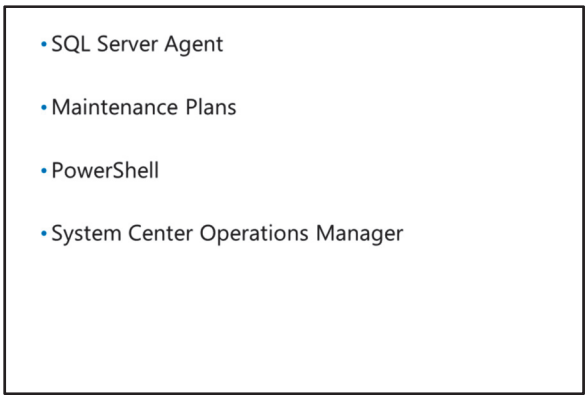
After you automate routine tasks, it is possible that their execution fails but no one notices. For example, an automated backup of databases may fail but this failure is not noticed until one of the backups is needed.

In addition to automating the routine tasks, you must ensure that you create notifications for when the tasks fail, even if you cannot imagine a situation where they might. For example, you may create a backup strategy that produces database backups in a given folder. The job might run reliably for years until another administrator inadvertently deletes or renames the target folder. You have to know as soon as this problem occurs, so that you can rectify the situation.

A proactive administrator will try to detect potential problems before they occur. For example, rather than receiving a notification that a job failed because a disk was full, an administrator might schedule regular checks of available disk space and make sure a notification is received when it is starting to get too low. SQL Server provides alerts on system and performance conditions for this type of scenario.

Available Options for Automating SQL Server Management

As a database administrator, you have certain tasks that should be performed regularly. Within the Microsoft ecosystem, an administrator has the choice of using several different technologies to help automate these regular tasks.



SQL Server Agent

SQL Server Agent is the primary option for administrators. It installed by default with an installation of SQL Server. This will therefore be the primary focus of this module.

Maintenance Plans

SQL Server provides a Maintenance Plan Wizard that prompts administrators to consider regular tasks to maintain the health of SQL Server. Behind the scenes, the wizard creates SQL Server Agent jobs and a SQL Server Integration Service (SSIS) package to orchestrate these jobs and manage workflow.

PowerShell

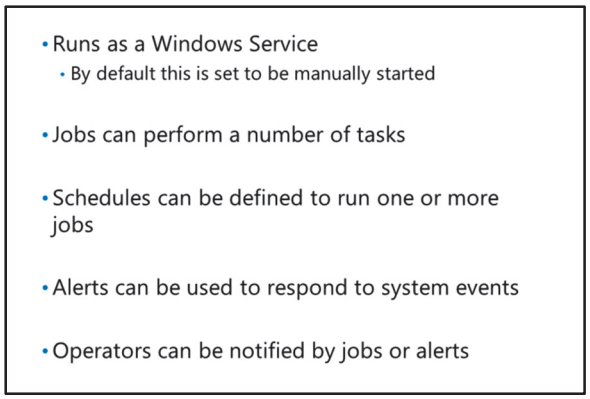
This command-line utility can be used to script, and therefore automate, many tasks inside and outside SQL Server. These scripts can be included as tasks in SQL Server Agent jobs, but also scheduled outside SQL Server to perform more complicated operations.

System Center Operations Manager (SCOM)

SCOM is an enterprise level suite of tools with which administrators can monitor servers, including SQL Servers, across an organization. Primarily focused on performance monitor, reporting, and notifications, SQL Server Agent jobs can write to logs that are monitored by SCOM, and then become part of operational level reporting.

Overview of SQL Server Agent

The primary method for automating the management, administration, and other routine tasks when working with SQL Server is to use the SQL Server Agent. The SQL Server Agent runs as a Windows service and has to be running constantly to perform its roles of executing jobs, firing alerts, and contacting operators. Because of this, you should configure it to begin automatically when the operating system starts. The default setting after installation of SQL Server is for SQL Server Agent to be started manually, so you must change this before working with it.



You can configure the start mode for SQL Server Agent in the properties of the SQL Server Agent service in SQL Server Configuration Manager. There are three available start modes:

- **Automatic**. The service begins when the operating system starts.
- **Disabled**. The service will not start, even if you attempt to do so manually.
- **Manual**. The service needs to be started manually.

You can also configure the SQL Server Agent service to restart automatically, if it stops unexpectedly, by using the properties page for the SQL Server Agent in SQL Server Management Studio (SSMS). To restart automatically, the SQL Server Agent service account must be a member of the local administrators group for the computer where SQL Server is installed—but this is not considered a best practice. A better option would be to use an external monitoring tool such as SCOM to monitor and restart the SQL Server Agent service if necessary.

The SQL Server Agent supplies a management framework that is based on four core object types:

- Jobs that you can use to automate tasks.
- Schedules to set the frequency and time to run jobs.
- Alerts that you can use to respond to events.
- Operators that define administrative contacts for alerts.

Jobs

You can use jobs to execute command-line scripts, Windows PowerShell® scripts, Transact-SQL scripts, SQL SSIS packages, and so on. You can also use them to schedule a wide variety of task types, including tasks involved in the implementation of other SQL Server features. These include replication, Change Data Capture (CDC), Data Collection, and Policy Based Management (PBM).

 **Note:** Replication, CDC, and PBM are advanced topics that are beyond the scope of this course.

Schedules

Schedules specify when, and how, a job runs. It's a many-to-many relationship, in that a schedule can run many jobs, and a job can be included in many schedules.

Alerts

The alert system provided by SQL Server Agent is capable of responding to a wide variety of alert types, including SQL Server error messages, SQL Server performance counter events, and Windows Management Instrumentation (WMI) alerts.

Operators

You can configure an action to happen in response to an alert, such as the execution of a SQL Server Agent job or sending a notification to an administrator. In SQL Server Agent, administrators that you can notify are called operators. One common way of notifying operators is by using Simple Mail Transfer Protocol (SMTP)-based email. (Alerts and operators are discussed later in this course.)

 **Note:** You can use other SQL Server features to automate complex monitoring tasks—for example, Extended Events—but this is beyond the scope of this course.

Demonstration: SQL Server Agent

In this demonstration, you will see how to:

- Create a simple job, and a step.
- Run a job, and review the output.

Demonstration Steps

Create a Job

1. Ensure that the **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. In File Explorer, navigate to **D:\Demofiles\Mod08**, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. On the taskbar, click **Microsoft SQL Server Management Studio**.
5. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, and then click **Connect**.
6. In Object Explorer, expand **SQL Server Agent**, and then expand **Jobs**.

7. Note that there are existing jobs on the server.
8. Right-click the **Jobs** folder, click **New Job**.
9. In the **New Job** dialog box, in the **Name** box, type **Manual AdventureWorks Backup**, and in the **Category** list, click **Database Maintenance**.
10. On the **Steps** page, click **New**.
11. In the **New Job Step** dialog box, in the **Step name** box, type **Backup AdventureWorks**, and in the **Database** list, click **AdventureWorks**.
12. In the **Command** box, type the following, and then click **OK**:

```
BACKUP DATABASE [AdventureWorks] TO  DISK =
N'D:\Demofiles\Mod08\Backups\AdventureWorksAgentBackup.bak' WITH NOFORMAT, INIT,
NAME = N'AdventureWorks-Full Database Backup', COMPRESSION,  STATS = 10
GO
```

13. In the **New Job** dialog box, click **OK**.
14. In Object Explorer, under **Jobs**, note that the new job, **Manual AdventureWorks Backup** is displayed.

Run and Review the Output from a Job

1. In Object Explorer, under **Jobs**, right-click **Manual AdventureWorks Backup**, and then click **Start Job at Step**.
2. In the **Start Jobs - MIA-SQL** dialog box, note that the status of each step when it finishes changes to **Success**, and then click **Close**.
3. In Object Explorer, under **SQL Server Agent**, double-click **Job Activity Monitor**.
4. In the **Job Activity Monitor - MIA-SQL** window, review the information available for SQL Server Agent jobs.
5. In the **Agent Job Activity** table, review the information in the **Manual AdventureWorks Backup** row, and then click **Close**.
6. In File Explorer, navigate to **D:\Demofiles\Mod08\Backups**, and verify that the **AdventureWorksAgentBackup.bak** file has been created.
7. In SSMS, in the **Job Activity Monitor - MIA-SQL** window, click **Close**.
8. Leave SSMS open for the next demonstration.

Categorize Activity

What are the four core objects types provided by SQL Server Agent?

Items	
1	Jobs
2	Maintenance Plans
3	Schedules
4	Backup Tasks
5	Alerts
6	Logs
7	Operators
8	SCOM Reporting

Category 1		Category 2
Provided by SQL Server Agent		Not provided by SQL Server Agent

Lesson 2
Working with SQL Server Agent

Because SQL Server Agent is the primary tool for automating tasks within SQL Server, database administrators must be proficient at creating and configuring SQL Server Agent jobs. You can create jobs to implement a variety of different types of task and categorize them for ease of management.

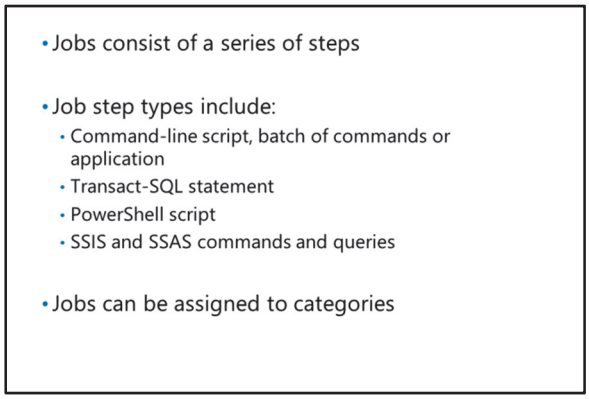
In this lesson, you will learn how to create, schedule, and script jobs.

Lesson Objectives

After completing this lesson, you will be able to:

- Define jobs, job types, and job categories.
- Create job steps.
- Schedule jobs for execution.
- Script jobs.

Defining Jobs, Job Types, and Job Categories



Creating a job involves putting together a series of steps that the job will execute, along with the workflow that determines which steps should be executed—and in which order. In most jobs, the steps will execute in sequential order, but you can control the order if required. For example, you may want one step to execute if the previous one succeeded, but a different step to execute if the previous one failed.

After you determine the steps, you have to decide when to execute the job. You will likely run most of your SQL Server Agent jobs on defined schedules. SQL Server helps you to create a flexible set of schedules that you can share between jobs.

It is important to learn to script jobs that have been created so that, if a failure occurs, you can quickly recreate the job and reconstruct it in other environments. For example, you may create your jobs in a test environment, but then want to move them to your production environment.

Job Step Types

Every job step has an associated type that defines which operation to run. The most commonly-used types include:

- Executing a command-line script, batch of commands, application, or operating system commands.
- Executing a Transact-SQL statement.
- Executing a Windows PowerShell script.
- Executing SQL Server Integration Services and Analysis Services commands and queries.

Note: While the ability to execute ActiveX® scripts is retained for backwards compatibility, this option is deprecated and you should not use it for new development.

Creating Jobs

You can use SSMS to create jobs or you can execute the **sp_add_job** system stored procedure, in addition to other system stored procedures, to add steps and schedules to the job. After you create the job, SQL Server stores the job definition in the **msdb** database, alongside all the SQL Server Agent configuration.

This example shows how to create a simple job in Transact-SQL. This only creates a job; a job step will be added in the next topic.

Using sp_add_job

```
USE msdb;
GO

EXEC sp_add_job
    @job_name = 'HR database backup',
    @enabled = 1,
    @description = 'Backup the HR database',
GO
```

A third option is to make use of the SQL Server Management Objects, and call the **Create** method from C# or PowerShell.

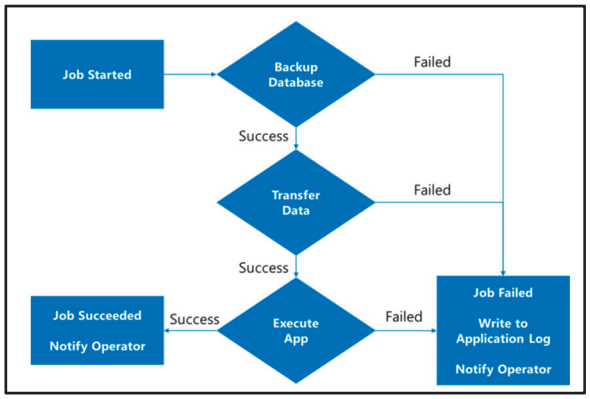
Job Categories

You can organize your jobs into categories either by using the SQL Server built-in categories, such as Database Maintenance, or by defining your own.

This is useful when you have to perform actions that are associated with jobs in a specific category. For example, you could create a job category called SQL Server Policy Check and write a PowerShell script to execute all the jobs in that category against your SQL Server servers.

Creating Job Steps

After you create a job, you can add job steps to it by using SSMS or by executing the **sp_add_jobstep** system stored procedure. The information in the following table describes the key arguments of the stored procedure:



Argument	Description
@job_id	Unique identification number of job to which to add the step.
@job_name	User-friendly name of job to which to add the step.

Argument	Description
@step_id	Unique identification number of the job step, starting at 1 and incrementing by 1.
@step_name	User-friendly name of job step.
@subsystem	Subsystem for SQL Server Agent to use to execute the command (for example, TSQL, Dts, CMDEXEC or PowerShell). TSQL is the default value.
@command	Command to execute. SQL Server Agent provides token substitution that gives you the same flexibility that variables provide when you write software programs.
@on_success_action	Action to perform if the step succeeds (for example, quit or go to the next step).
@on_fail_action	Action to perform if the step fails (for example, quit or go to the next step).
@retry_attempts	Number of retry attempts if the step fails.
@retry_interval	Time to wait between retry attempts.

For a full list of the available parameters see *sp_add_jobstep (Transact-SQL)* in Microsoft Docs:

sp_add_jobstep (Transact-SQL)

http://aka.ms/Cmljv9

When successful, SQL Server advances by default to the next job step, and stops when a job step fails. However, job steps can continue with any step defined in the job, using the success or failure flags. By configuring the action to occur on the success and failure of each job step, you can create a workflow that determines the overall logic flow of the job. Note that, in addition to each job step having a defined outcome, the overall job reports an outcome. This means that, even though some job steps may succeed, the overall job might still fail.

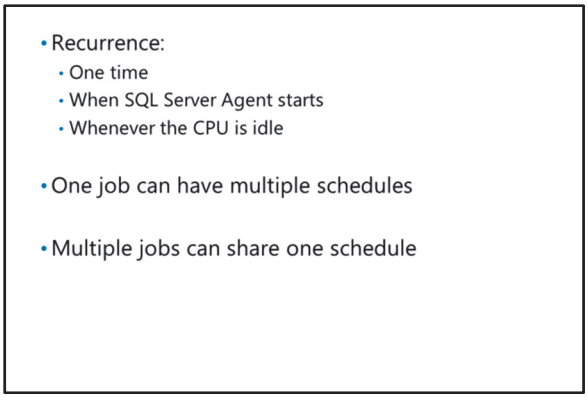
You can specify the number of times that SQL Server should attempt to retry execution of a job step if the step fails. You can also specify the retry intervals (in minutes). For example, if the job step requires a connection to a remote server, you could define several retry attempts in case the connection fails.

This example shows how to add a job step to a job:

Using sp_add_jobstep

```
USE msdb;
GO

EXEC sp_add_jobstep
    @job_name = 'HR database backup',
    @step_name = 'Set HR database to read only',
    @subsystem = 'TSQL',
    @command = 'ALTER DATABASE HR SET READ_ONLY',
    @retry_attempts = 2,
    @retry_interval = 2;
GO
```

Scheduling Jobs for Execution



You can define one or more schedules for every job and use them to start jobs at requested times. In addition to the standard recurring schedules, you can also assign a number of special recurrence types:

- One-time execution.
- Start automatically when SQL Server Agent starts.
- Start whenever the CPU becomes idle.

When the recurrence pattern for a job is complex, you might have to create multiple schedules. While you can share each schedule between several jobs, you should avoid having too many starting at the same time.

Even though a job might have multiple schedules, SQL Server will limit it to a single concurrent execution. If you try to run a job manually while it is running as scheduled, SQL Server Agent refuses the request. Similarly, if a job is still running when it is scheduled to run again, SQL Server Agent refuses to let it do so.

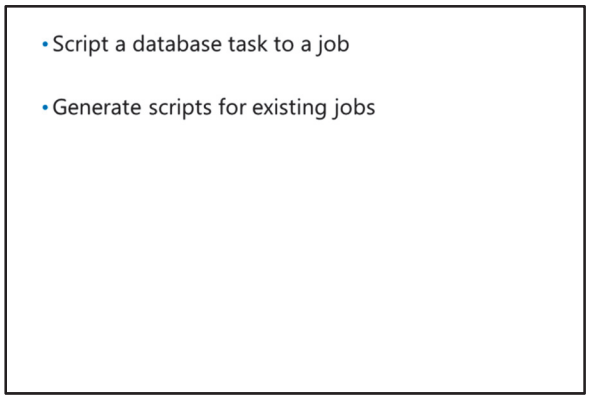
The following example shows how to create and attach a schedule for Shift 1 to the job created in earlier examples:

Using sp_add_schedule and sp_attach_schedule

```
USE msdb;
GO

EXEC sp_add_schedule
    @schedule_name = 'Shift 1',
    @freq_type = 4,
    @freq_interval = 1,
    @freq_subday_type = 0x8,
    @freq_subday_interval = 1,
    @active_start_time = 080000,
    @active_end_time = 170000;
GO


EXEC sp_attach_schedule
   @job_name = 'HR database backup',
   @schedule_name = 'Shift 1';
GO
```

Scripting Jobs

There are two approaches to scripting jobs. The first is to script a standard database action as a new job; the second is to script existing jobs. The easiest way to complete these approaches is by using SSMS, although you can complete these tasks by using PowerShell.



Script a Task to a Job

In SSMS, you can script tasks to jobs. You can script database backups, rebuilding indexes, or drop and recreate tables. When in a database task, look for the scripting button, which offers the following options:

- Script Action to a New Query Window.
- Script Action to File.
- Script Action to Clipboard.
- Script Action to Job.

If the last option is selected, the New Job dialog is opened and prepopulated with the current task as the first step in the job.

Generate Scripts for Existing Jobs

After creating a job, you can script it to a file, or new query window, in a similar way to tables and stored procedures in a database. A business reason to do this might include replicating these jobs onto another server, or backing up the jobs in version control software. The options available for jobs are to script as CREATE statements, DROP statements, or DROP and CREATE statements. You will see how to perform these in the following demonstration.

Demonstration: Scripting Jobs

In this demonstration, you will see how to:

- Script a task to a job.
- Generate scripts for existing jobs.

Demonstration Steps

Script a Task to a Job

1. In SSMS, in Object Explorer, expand **Databases**, right-click **AdventureWorks**, point to **Tasks**, and then click **Back Up**.
2. In the **Back Up Database - AdventureWorks** dialog box, in the **Destination** section, click the existing backup destination, click **Remove**, and then click **Add**.
3. In the **Select Backup Destination** dialog box, in the **File name** box, type **D:\Demofiles\Mod08\Backups\AdventureWorksScript.bak**, and then click **OK**.
4. In the **Back Up Database - AdventureWorks** dialog box, on the toolbar, in the **Script** drop-down list, select **Script Action to Job**.

5. In the **New Job** dialog box, on the **General** page, note the default name for the job (Back Up Database - AdventureWorks).
6. On the **Steps** page, note that the job includes one **Transact-SQL step** named **1**.
7. On the **Schedules** page, click **New**.
8. In the **New Job Schedule** dialog box, in the **Name** box, type **Week Days**.
9. In the **Frequency** section, select the **Monday**, **Tuesday**, **Wednesday**, **Thursday**, and **Friday** check boxes, clear the **Sunday** check box, and then click **OK**.
10. In the **New Job** dialog box, click **OK**.
11. In the **Back Up Database - AdventureWorks** dialog box, click **Cancel** so that the job is saved, but not yet run.
12. In Object Explorer, verify that the **Back Up Database - AdventureWorks** job appears in the **Jobs** folder.

Generate Scripts for Existing Jobs

1. In Object Explorer, under **Jobs**, right-click **Check AdventureWorks DB**, point to **Script Job as**, point to **CREATE To**, and then click **New Query Editor Window**. This generates the Transact-SQL code necessary to create the job.
2. In Object Explorer, right-click **Back Up Database - AdventureWorks**, point to **Script Job as**, point to **CREATE To**, and then click **Clipboard**.
3. Place the insertion point at the end of the Transact-SQL code in the query editor window, and then on the **Edit** menu, click **Paste**.
4. Save the Transact-SQL script as **Create Jobs.sql** in the **D:\Demofiles\Mod08** folder.
5. This technique is useful to generate script creation jobs so that they can be recreated if they are accidentally deleted or are required on a different server.
6. Keep SQL Server Management Studio open for the next demonstration.

Sequencing Activity

Put the following SQL Server Agent steps in the order required to create a job, by numbering each to indicate the correct order.

	Steps
	Create a job.
	Categorize the job.
	Create one or more steps.
	Create one or more schedules.
	Create or select an operator for notifications.
	Schedule the job for execution.

Lesson 3
Managing SQL Server Agent Jobs

When you automate administrative tasks, you must ensure that they execute correctly. To help with this, SQL Server writes entries to history tables in the **msdb** database on the completion of each job.

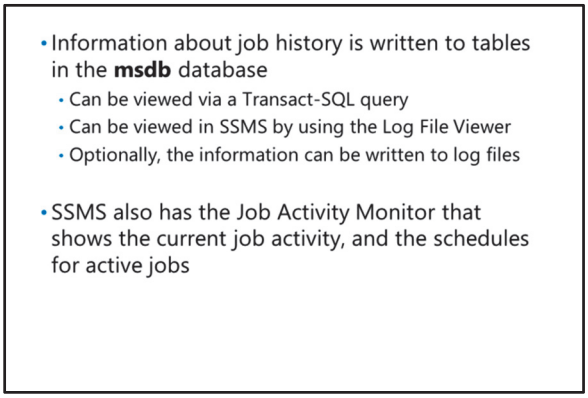
In this lesson, you will learn how to query the history tables, and how to troubleshoot any issues that may occur.

Lesson Objectives

After completing this lesson, you will be able to:

- View job history.
- Query SQL Server Agent-related system tables and views.
- Troubleshoot failed jobs.

Viewing Job History



Every job and job step has an outcome. If any one of them fails, you have to find out why and rectify the issue before the job or step runs again. SQL Server Agent tracks the outcomes of jobs and their steps in system tables in the **msdb** database. You can also choose to write this information to the Windows Application log or SQL Server log.

You can view the history for each job by using SSMS: right-click on the job and then click **View History**. The same information is also available by querying the job history tables. By default, the most recent 1,000 entries of job history are retained; however, you can configure this retention policy to base it instead on age or size of data. You'll see how to change this setting in the demonstration at the end of this lesson.

The Object Explorer window in SSMS also provides a Job Activity Monitor. This displays a view of currently executing jobs and data showing the results of the previous execution, along with the scheduled time for the next execution of the job.

Querying SQL Server Agent-related System Tables and Views

There are many SQL Server Agent-related system tables and views that you can query to retrieve information about jobs, alerts, schedules, and operators. All the tables are stored in the **msdb** database in the **dbo** schema. For example, job history is held in the **dbo.sysjobhistory** table, and there is a list of jobs in the **dbo.sysjobs** table.

The following example shows code querying the **dbo.sysjobs**, **dbo.sysjobhistory**, and **dbo.sysjobsteps** tables to retrieve information about an enabled job, and how long each step ran:

- Configuration and history information is stored in **msdb.dbo** schema
- Use history tables to automate collection of job history over several systems
- The **dbo.sysjobhistory** table contains a row for each job with a **step_id** = 0. This contains the result for the whole job

Querying Job History Tables

```
SELECT jobs.name AS 'Job Name', jobsteps.step_id AS 'Step', jobsteps.step_name AS 'Step
Name',
 msdb.dbo.agent_datetime(run_date, run_time) AS 'Run Date',
 ((run_duration/10000*3600 + (run_duration/100)%100*60 + run_duration%100 + 31 ) / 60)
         AS 'Duration'
FROM msdb.dbo.sysjobs jobs
INNER JOIN msdb.dbo.sysjobsteps jobsteps
  ON jobs.job_id = jobsteps.job_id
INNER JOIN msdb.dbo.sysjobhistory history
  ON jobsteps.job_id = history.job_id
  AND jobsteps.step_id = history.step_id
  AND history.step_id <> 0
WHERE jobs.enabled = 1
ORDER BY 'Job Name', 'Run Date' desc;
```

Note: The INNER JOIN on the **dbo.sysjobhistory** table has the clause **history.step_id** <> 0. The history table has entries where the step_id = 0. These rows contain the overall outcome of a job. By returning all rows where step_is <> 0, the previous query returns information about each step of the job.

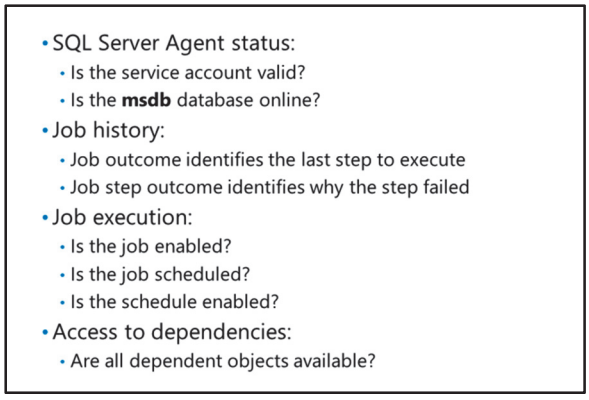
For more information about the system tables that store SQL Server Agent data, see *SQL Server Agent Tables (Transact-SQL)* in Microsoft Docs:

SQL Server Agent Tables (Transact-SQL)

http://aka.ms/Hqcjrg

Troubleshooting Failed Jobs

Jobs do not always execute as expected and will sometimes fail to execute at all. It is important to follow a consistent process when attempting to work out why a job is failing. There are four basic steps for troubleshooting jobs—checking SQL Server Agent status, reviewing job history, checking job execution, and checking access to dependencies.



Checking SQL Server Agent Status

If SQL Server Agent is not running, no jobs can run. Make sure the service is set to start automatically and, if jobs are failing, attempt to start it manually. If the service will still not start, check the following:

- That the service account for the service is valid, that the password for the account has not changed, and that the account is not locked out. If any of these checks are the issue, the service will not start and details about the problem will be written to the computer's system event log.
- That the **msdb** database is online. If the **msdb** database is corrupt, suspect, or offline, SQL Server Agent will not start.

Reviewing Job History

Review the job outcome to identify the last step run. If the job was unsuccessful because a job step failed—the most common situation—the error of the job step cannot be seen at this level. You need to review the individual job step outcome for the failed job.

Checking Job Execution

If SQL Server Agent is running but an individual job will not run, check the following items:

- That the job is enabled. Disabled jobs will not run.
- That the job is scheduled. The schedule may be incorrect or the time for the next scheduled execution may be in the future.
- That the schedule is enabled. Both jobs and schedules can be disabled and a job will not run on a disabled schedule.

Checking Access to Dependencies

Verify that all dependent objects in the job, such as databases, files, and procedures, are available. Jobs often run in a different security context to the user who creates them. Incorrect security settings present a common problem that causes job execution to fail.

Demonstration: Viewing Job History and Resolving Failed Jobs

In this demonstration, you will see how to:

- Run jobs.
- Troubleshoot a failed job.

Demonstration Steps

Run Jobs

1. In SQL Server Management Studio, in Object Explorer, right-click **Back Up Database - AdventureWorks**, and then click **Start Job at Step**.
2. When the job has completed successfully, click **Close**.
3. In Object Explorer, right-click **Check AdventureWorks DB**, and then click **Start Job at Step**.
4. Note that this job does not start automatically because it has more than one job step.
5. In the **Start Job on 'MIA-SQL'** dialog box, in the **Start execution at step** table, click **1**, and then click **Start**. Note that the job fails, and then click **Close**.

Troubleshoot a Failed Job

1. In Object Explorer, right-click **Back Up Database - AdventureWorks**, and then click **View History**.
2. In the **Log File Viewer - MIA-SQL** dialog box, expand the date for the most recent instance of the job, note that all steps succeeded, and then click **Close**.
3. In Object Explorer, right-click **Check AdventureWorks DB**, and then click **View History**.
4. In the **Log File Viewer - MIA-SQL** dialog box, expand the date for the most recent failed instance of the job, and note that the step **3** failed.
5. Select the step that failed, and in the pane at the bottom of the dialog box, view the message that was returned. You may have to scroll to the bottom. Then click **Close**.
6. In Object Explorer, double-click **Check AdventureWorks DB**.
7. In the **Job Properties - Check AdventureWorks DB** dialog box, on the **Steps** page, in the **Job step list** table, click step **3**, and then click **Edit**.
8. In the **Job Step Properties - Check DB** dialog box, click **Parse**.
9. In the **Parse Command Text** dialog box, note the same error message that was shown in Job History, and then click **OK**.
10. In the **Job Step Properties - Check DB** dialog box, modify the text in the **Command** box as follows, and then click **OK**:

```
DBCC CHECKDB ('AdventureWorks');
```

11. In the **Job Properties - Check AdventureWorks DB** dialog box, click **OK**.
12. In Object Explorer, right-click **Check AdventureWorks DB**, and then click **Start Job at Step**.
13. In the **Start Job on 'MIA-SQL'** dialog box, in the **Start execution at step** table, click **1**, and then click **Start**.
14. When the steps complete successfully, click **Close**.
15. In **Object Explorer**, double-click **Job Activity Monitor**.

16. In the **Job Activity Monitor - MIA-SQL** dialog box, in the **Agent Job Activity** table, note the status of the **Check AdventureWorks DB** job, and then click **Close**.
17. In the **D:\Demofiles\Mod08\AdventureWorks** folder, view the text files generated by the job.
18. In the **D:\Demofiles\Mod08\Backups** folder, verify that a backup file was created by the **Back Up Database - AdventureWorks** job.
19. Keep SQL Server Management Studio open for the next demonstration.

Categorize Activity

What are four steps that can be undertaken to troubleshoot failed jobs?

Items	
1	Check SQL Server Agent Status
2	Start and Stop SQL Server
3	Review Job History
4	Check Free Disk Space
5	Check Job Execution
6	Check Activity Monitor
7	Check Access to Dependencies

Category 1		Category 2
Troubleshooting step		Other database activity

Lesson 4
Multiserver Management

You may have jobs that run across multiple servers that would benefit from being automated. SQL Server provides multiserver administration functionality for you to distribute jobs across your enterprise.

In this lesson, you will learn about the concepts behind multiserver administration and how to implement jobs across multiple servers.

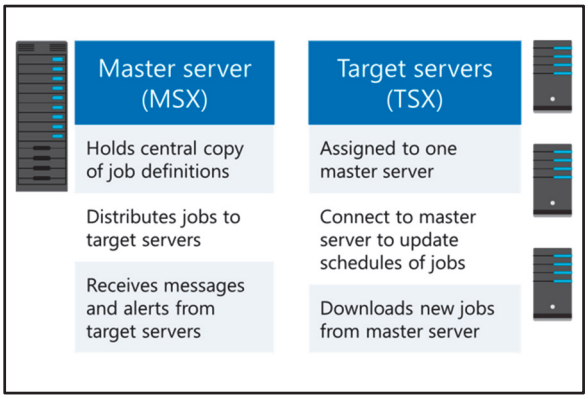
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the concepts of multiserver administration.
- Explain the considerations for multiserver administration.
- Run jobs on target servers.
- Automate multiserver administration.

Multiserver Management Concepts

Multiserver administration involves one master server that stores the master copy of the jobs and distributes them to one or more target servers. The master server can also receive events from the target servers that update it with the status of the jobs they run. The target servers are assigned to one master server, to which they periodically connect, to update their schedule of jobs and download any new ones.

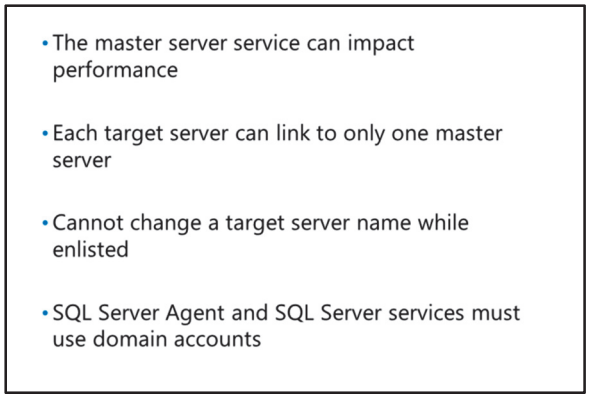


For example, you can create a backup job on a central server, distribute it to all servers in your organization, and then monitor the status of all the jobs from that central server.

It is recommended that if you have a large number of target servers, you should avoid defining your master server on a production server that has significant performance requirements. This is because target server traffic may slow performance on the production server.

Considerations for Multiserver Management

You must take into account a number of considerations before setting up multiserver administration:



- Because the master server both distributes jobs to other servers and receives events from them, the role can impact a high-load performance server. Consider creating the master server on a server that does not have a significant workload or network traffic requirement.
- Each target server can only link to one master server. If you want to change your master server, you must first defect all target servers from the master server, and then enlist them all to the new master server.
- The master server uses the name of a target server. Therefore, if you want to change the name of the target server, you must first defect it from the master server, rename it, and then enlist it back to the master server.
- Because they have to communicate across the multiple servers, the SQL Server Agent service and SQL Server service must run using Windows domain accounts.

Demonstration: Configuring Master and Target Servers

In this demonstration, you will see how to:

- Use the Master Server Wizard.
- Use Transact-SQL to register target servers.

Demonstration Steps

Use the Master Server Wizard

1. In SQL Server Management Studio, in Object Explorer, under **MIA-SQL**, right-click **SQL Server Agent**, point to **Multi Server Administration**, and then click **Make this a Master**.
2. In the **Master Server Wizard - MIA-SQL** dialog box, on the **Welcome to the Master Server Wizard** page, click **Next**.
3. On the **Master Server Operator** page, in the **E-mail address** box, type **student@adventureworks.com**, and then click **Next**.
4. On the **Target Servers** page, expand **Database Engine**, expand **Local Server Groups**, click **mia-sql\sql2**, click the **>**, and then click **Next**.
5. In the **Checking Server Compatibility** dialog box, click **Close**.
6. On the **Master Server Login Credentials** page, click **Next**.
7. On the **Complete the Wizard** page, click **Finish**.
8. When configuration is complete, click **Close**.

Use Transact-SQL to Register a Target Server

1. In Object Explorer, on the toolbar, click **Connect**, and then click **Database Engine**.
2. In the **Connect to Server** dialog box, in the **Server name** list, select **MIA-SQL\SQL3**, and then click **Connect**.
3. In Object Explorer, under **MIA-SQL\SQL3**, expand **Databases**, expand **System Databases**, right-click **msdb**, and then click **New Query**.
4. In the new query window, type the following command, and then click **Execute**:

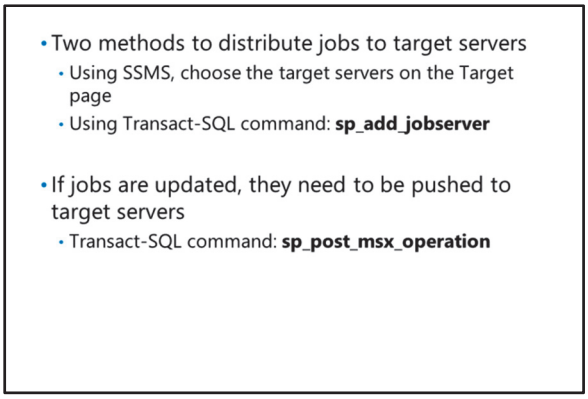
```
EXEC dbo.sp_msx_enlist N'MIA-SQL';
```

5. In Object Explorer, under **MIA-SQL**, right-click **SQL Server Agent (MSX)**, point to **Multi Server Administration**, and then click **Manage Target Servers**.
6. In the **Target Server Status - MIA-SQL** dialog box, note that both **MIA-SQL\SQL2** and **MIA-SQL\SQL3** are listed as target servers.
7. In the **Target Server Status - MIA-SQL** dialog box, click **Close**.
8. Close SSMS without saving any changes.

Running Jobs on Target Servers

After you configure the master and target servers, you can begin to distribute jobs from one to the other by using SSMS or Transact-SQL. On the master server, in SSMS, in the **Properties** dialog box for a job, you can choose to target either the local server or multiple servers, selecting which ones to run the job on.

A job can only have a target of either running locally, or on target servers. To use Transact-SQL to add a job to a target server, that job must be defined as a multiserver job.



To distribute jobs to target servers by using Transact-SQL, use the **sp_add_jobserver** statement as shown in the following example:

Distributing Jobs to Target Servers

```
USE msdb;
GO

EXEC dbo.sp_add_jobserver @job_name='Backup master database', @server_name='MIA-
SQL\SQL3';
GO
```

If you change the definition of a multiserver job after distributing it to the target servers, you have to ensure that SQL Server adds the change to the download list for the target servers to be updated.

You can do this by executing the **sp_post_msx_operation** stored procedure as shown in the following example:

Updating Distributed Jobs

```
USE msdb;
GO

sp_post_msx_operation @operation='INSERT', @object_type='JOB', @job_id='<job_id>';
GO
```

You only have to execute this code when you add, update, or remove job steps or schedules; **sp_update_job** and **sp_delete_job** automatically add entries to the download list.

Note: You can locate the **job_id** property by querying the **dbo.sysjobhistory** or **dbo.sysjobs** tables or by viewing the job properties in SSMS.

Automating Multiserver Maintenance

Along with creating jobs that can be run on multiple servers, you can define maintenance plans that can run in a multiserver environment—the primary difference being that a maintenance plan can run locally and on target servers. This isn't possible for SQL Server Agent jobs.

Maintenance plans can be created manually, using an interface similar to that used to create SSIS packages. You can create complex workflow that supports routing logic, depending on whether steps succeed or fail. For simple plans, SSMS provides a wizard to walk through the creation of maintenance plans.

- Maintenance plans support creating SQL Server Agent jobs that can run on master and target servers
- The execution and monitoring is all undertaken on the master server
- The Maintenance Plan Wizard can create simple maintenance plans

Using the Maintenance Plan Wizard

The Maintenance Plan Wizard in SSMS guides you through building a plan with a predefined set of actions. You can also use the wizard to select the target servers for the plan. These plans can then can be scheduled to execute across multiple servers. The actions the wizard supports are:

- Checking database integrity.
- Shrinking databases.
- Reorganizing indexes.
- Rebuilding indexes.
- Updating statistics.
- Cleaning up history.
- Databases backups.
- Running previously created SQL Server Agent jobs.

After completing the wizard, you will see the SQL Server Agent jobs and schedules the maintenance plan has created in the Object Explorer in SSMS.

The advantage of this approach is that you can define a set of actions once, and then run those same steps across your whole organization—with all the executing, monitoring and reporting of these jobs being undertaken on one master server.

Check Your Knowledge

Question	
Which of the following statements is false?	
Select the correct answer.	
	A master server can have multiple target servers.
	A high-load master server won't have any adverse performance impact by having a number of target servers.
	Each target server can only connect to a single master server.
	Changing the name of a target server requires it to be registered with the master server.

Lab: Automating SQL Server Management

Scenario

You are a database administrator (DBA) at Adventure Works Cycles, with responsibility for databases on the MIA-SQL instance of SQL Server. Routine tasks that must be performed on this instance have previously been performed manually, but you now plan to automate these tasks using SQL Server Agent.

Objectives

After completing this lab, you will be able to:

- Create SQL Server Agent jobs.
- Test jobs.
- Schedule jobs.
- Configure master and target servers.

Estimated Time: 75 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Create a SQL Server Agent Job

Scenario

The **HumanResources** database must be backed up every day. Additionally, after the backup has been created, the backup file must be copied to a folder, which is automatically replicated to a cloud service for offsite storage of various backup files.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create a Backup Job

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab08\Starter** folder as Administrator.

▶ Task 2: Create a Backup Job

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance of SQL Server.
2. Create a job named **Backup HumanResources** on the MIA-SQL instance of SQL Server.
3. Add a Transact-SQL step that runs in the **HumanResources** database and executes the following command. The output from the command should be saved as a text file in the D:\ folder:

```
BACKUP DATABASE HumanResources TO DISK = 'D:\HumanResources.bak';
```

4. Add an operating system command step that runs the following command to move the backup file to the **D:\Labfiles\Lab08\Starter** folder:

```
Move D:\HumanResources.bak D:\Labfiles\Lab08\Starter\HumanResources.bak /Y
```

5. Ensure that the job is configured to start with the Transact-SQL backup step.
6. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have created a job named **Backup HumanResources**.

Exercise 2: Test a Job

Scenario

You plan to test the new backup job before you schedule it to execute in a production environment. Through the process of testing, you identify a defect in the job. You must resolve the defect and ensure the backup is copied to the correct folder location.

The main tasks for this exercise are as follows:

1. Test the Backup Job
2. Investigate the Error
3. Resolve the Error and Retest

▶ Task 1: Test the Backup Job

1. Run the **Backup HumanResources** job that you created in the previous exercise.
2. Note that the **Start Jobs - MIA-SQL** dialog shows an error.

▶ Task 2: Investigate the Error

1. View the history of the **Backup HumanResources** job.
2. Investigate why Step 2 of the job failed.

 **Note:** Hint: You may have to expand the log file row to see the details of each step executed.

3. Find the error message detail:

```
Executed as user: ADVENTUREWORKS\ServiceAcct. The syntax of the command is incorrect.
Process Exit Code 1.  The step failed.
```

▶ Task 3: Resolve the Error and Retest

1. Resolve the syntax error you identified in the previous task by changing the second job step command to the following:

```
Move /Y D:\HumanResources.bak D:\Labfiles\Lab08\Starter\HumanResources.bak
```

2. Rerun the **Backup HumanResources** job.
3. Check that the **HumanResources.bak** file has been moved to the correct folder **D:\Labfiles\Lab08\Starter**.
4. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have tested the SQL Server Agent job and confirmed that it executes successfully.

Exercise 3: Schedule a Job

Scenario

You have created a job to back up the **HumanResources** database. Now you must schedule the job to run automatically each day.

The main tasks for this exercise are as follows:

1. Add a Schedule to the Job

2. Verify Scheduled Job Execution

▶ Task 1: Add a Schedule to the Job

1. Turn on the clock icon in the notification area of the taskbar so that you can easily see the current system time on the MIA-SQL virtual machine.
2. Add a schedule to the **Backup HumanResources** job so that the job runs every day, two minutes from the current system time.
3. Wait for the scheduled time, and then proceed with the next task.

▶ Task 2: Verify Scheduled Job Execution

1. Use the **Job Activity Monitor** to view the status of the **Backup HumanResources** job.
2. When the job is idle, verify that the **Last Run Outcome** for the job is **Succeeded**, and that the **Last Run** time is the time that you scheduled previously.
3. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have created a schedule for the **Backup HumanResources** job.

Exercise 4: Configure Master and Target Servers

Scenario

You have been tasked with backing up all the system databases within your company. You have decided to utilize the ability to create a maintenance plan and run this plan across all the company's servers.

The main tasks for this exercise are as follows:

1. Configure the Master and Target Servers

2. Create a Maintenance Plan

3. Monitor the Job History

▶ Task 1: Configure the Master and Target Servers

1. Use SQL Server Agent to register **MIA-SQL** as the master server.
2. Register **MIA-SQL\SQL2** and **MIA-SQL\SQL3** as target servers.

▶ Task 2: Create a Maintenance Plan

1. Use the **Maintenance Plan Wizard** to create a new backup task called **System Database Backups** to run daily. Set the scheduled time to be five minutes from the current time.
2. Ensure all three servers are selected as target servers.

3. Set the destination of the backup files to be the **D:** drive, and select **create a sub-directory** for each database.
4. Set the destination for logging to the **D:** drive.
5. Wait five minutes to check the status of the job, by looking at the contents of the **D:** folder.

▶ Task 3: Monitor the Job History

1. Use the **Job Activity Monitor** on **MIA-SQL** and **MIA-SQL\SQL2**.
2. What are the differences in the history shown on the master and target servers?

Results: At the end of this exercise, you will have created and executed a maintenance plan across multiple servers.

Module Review and Takeaways

In this module, you have learned how to automate SQL Server Management by scheduling jobs, in addition to learning how to manage jobs across multiple instances of SQL Server.

 **Best Practice:** When using a large number of target servers, avoid defining your master server on a production server, because the target server traffic may impact performance on the production server.

Module 9

Configuring Security for SQL Server Agent

Contents:

Module Overview

Other modules in this course have demonstrated the need to minimize the permissions that are granted to users, following the principle of "least privilege." This means that users have only the permissions that they need to perform their tasks. The same logic applies to the granting of permissions to SQL Server Agent. Although it is easy to execute all jobs in the context of the SQL Server Agent service account, and to configure that account as an administrative account, a poor security environment would result from doing this. It is important to understand how to create a minimal privilege security environment for jobs that run in SQL Server Agent.

Objectives

After completing this module, you will be able to:

- Explain SQL Server Agent security.
- Configure credentials.
- Configure proxy accounts.

Lesson 1
Understanding SQL Server Agent Security

SQL Server Agent is a powerful tool for scheduling and executing a diverse range of administrative and line-of-business tasks. However, to make best use of it, you must understand the security model that is used for SQL Server Agent jobs and tasks. Many of the tasks that are executed by SQL Server Agent are administrative in nature, but a number of other tasks are performed on behalf of users. The need to be able to execute a wide variety of task types leads to a requirement for flexible security configuration.

Jobs need to be able to access many types of objects. In addition to objects that reside inside SQL Server, jobs often need to access external resources, such as operating system files and folders. These operating system (and other) dependencies also require a configurable and layered security model, to avoid the need to grant too many permissions to the SQL Server Agent service account.

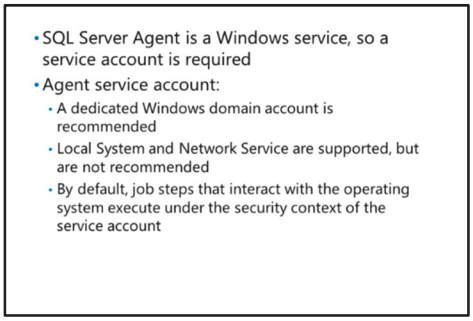
Lesson Objectives

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

- Give an overview of security in SQL Server Agent.
- Describe the fixed SQL Server Agent roles.
- Assign a security context to a SQL Server Agent job step.
- Troubleshoot problems with security in SQL Server Agent.

Overview of Security in SQL Server Agent

Like all services, the SQL Server Agent service has an identity within the Windows® operating system. The service startup account defines the Windows account in which the SQL Server Agent service runs. The account that is used defines the permissions that the SQL Server Agent service has when accessing network resources.



Agent Service Account

Although you can use the built-in Local System or Network Service account, it is preferable to create a dedicated Windows domain account to act as the service account for SQL Server Agent.

Note: The Local System account option is supported for backward compatibility only. For security reasons, the Network Service account option is also not recommended: Network Service has more capabilities than the service account requires. An account that has only the required permissions should be created and used instead.

A Windows domain account should be used and configured with the least possible privileges that will still allow operation. The account must be a member of the **sysadmin** fixed server role. It also requires the following Windows permissions:

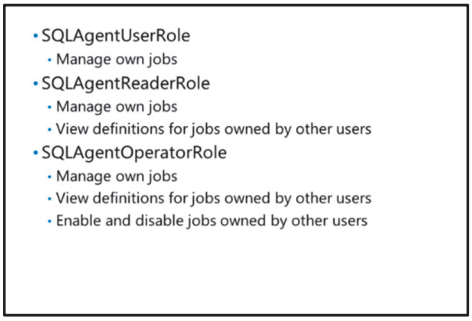
- Log on as a service
- By pass traverse checking
- Replace a process-level token
- Adjust memory quotas for a process
- Access the computer from the network

Note: The SQL Server Agent account cannot use SQL Server Authentication for its connection to the SQL Server Database Engine.

In many circumstances, SQL Server Agent jobs that run tasks that trigger Windows programs are executed in the security context of the service account. The options for configuring security for SQL Server Agent job steps are discussed later in this module.

SQL Server Agent Roles

As part of defining security for SQL Server Agent, you can control which logins have permission to manage SQL Server Agent jobs. All members of the **sysadmin** built-in server role have permission to administer SQL Server Agent jobs. If you need to grant permission to administer jobs to logins that are not members of the **sysadmin** role, three fixed roles are available for this purpose in the **msdb** database. From the lowest to the highest permission, these are:



- **SQLAgentUserRole**
- **SQLAgentReaderRole**
- **SQLAgentOperatorRole**

Each role has different permissions assigned to it, and roles that have greater privileges inherit the permissions of less privileged roles. For example, the **SQLAgentOperatorRole** role inherits the permissions of **SQLAgentUserRole** and **SQLAgentReaderRole**, in addition to the permissions that are assigned to it directly.

Note: To see and use the SQL Server Agent section of Object Explorer in SQL Server Management Studio, a user must be a member of one of the fixed SQL Server Agent roles, or a member of **sysadmin**.

SQLAgentUserRole

Members of **SQLAgentUserRole** have permission to edit, delete, execute, and view history of local jobs and job schedules of which they are the owner. They can also create new jobs and job schedules; the user will be the owner of any jobs and schedules that he or she creates.

Note: Remember that members of **SQLAgentUserRole** have permission to create and execute new jobs. Depending on the permissions that members of this role hold in other databases, there is the potential for them to affect server performance by creating and scheduling large numbers of jobs, or writing jobs that contain poorly performing Transact-SQL code.

SQLAgentReaderRole

Members of **SQLAgentReaderRole** have all of the permissions of **SQLAgentUserRole**; in addition, they can view job definitions, job schedule definitions, and job history for jobs that other users own—including multiserver jobs.

SQLAgentOperatorRole

Members of **SQLAgentOperatorRole** have all of the permissions of **SQLAgentUserRole** and **SQLAgentReaderRole**; in addition, they can enable and disable job definitions and job schedules that other users own.

For more information about the fixed SQL Server Agent roles, including a complete breakdown of the permissions of each role, see the topic *SQL Server Agent Fixed Database Roles* in Microsoft Docs:

SQL Server Agent Fixed Database Roles

http://aka.ms/M4weqy

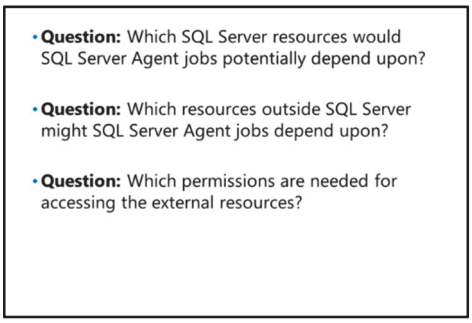
Discussion: Job Dependencies in SQL Server Agent

Discussion Topics

Question: Which SQL Server resources would SQL Server Agent jobs potentially depend upon?

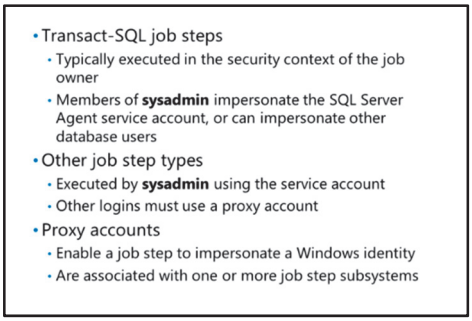
Question: Which resources outside SQL Server might SQL Server Agent jobs depend upon?

Question: Which permissions are needed for accessing the external resources?



Assigning Security Contexts to SQL Server Agent Job Steps

When you are planning SQL Server Agent jobs, you must consider the security context under which the steps in those jobs will be executed.



Transact-SQL Job Steps

Generally, when SQL Server Agent runs a Transact-SQL job, SQL Server Agent impersonates the job owner. However, if the owner of the job step is a member of the **sysadmin** fixed server role, the job step will run in the security context of the SQL Server Agent service, unless the **sysadmin** role chooses to have the step impersonate another user. Only members of the **sysadmin** fixed server role can specify that the job step should impersonate another user.

You specify an alternate database user name for a Transact-SQL job step in the **Run as user** box on the **Advanced** page of the **Job step properties** dialog box, or by using the *@database_user_name* parameter of the **sp_add_jobstep** and **sp_update_jobstep** system stored procedures.

Note: Although the **General** page of the **Job step properties** dialog box has a **Run As** box, this property is not used from Transact-SQL job steps.

Other Types of Job Step

For job step types that are not based on Transact-SQL, a different security model is used. All of the job step types other than Transact-SQL involve running a command-line executable file. Only members of the **sysadmin** fixed server role have direct permission to run job steps that interact with the command line. When a member of the **sysadmin** fixed server role runs a job step, it will execute by using the SQL Server Agent service account by default.

Note: As discussed earlier in the lesson, the range of permissions required to carry out different job steps can lead to the SQL Server Agent service account being highly privileged.

Proxy Accounts

As an alternative to using the SQL Server Agent account, you can use a proxy account to associate a job step with a Windows identity by using an object called a credential. You can associate a proxy account with one or more of the non-Transact-SQL subsystems that are used in job steps. Using proxy accounts means that you can use different Windows identities to perform the various tasks that are required in jobs. It provides better control of security by avoiding the need for a single account to have all of the permissions that are required to execute all jobs. Members of the SQL Server Agent fixed roles who are not members of **sysadmin** can be granted permission to use proxy accounts to execute job steps that are not based on Transact-SQL.

Creating and configuring credentials and proxy accounts is covered later in this module.

To select a proxy account for a step that is not based on Transact-SQL, you use the **Run As** box on the **General** page of the **Job step properties** dialog box, or the *@proxy_name* parameter of the **sp_add_jobstep** and **sp_update_jobstep** system stored procedures.

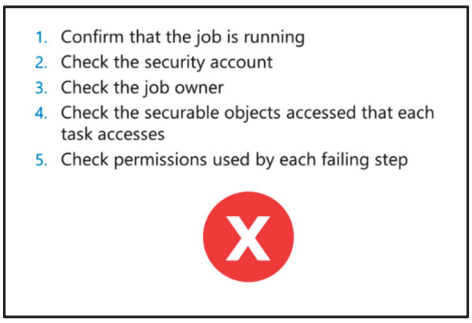
For more information about configuring job steps by using **sp_add_jobstep**, see the topic *sp_add_jobstep (Transact-SQL)* in Microsoft Docs:

sp_add_jobstep (Transact-SQL)

http://aka.ms/cmljv9

Troubleshooting Security in SQL Server Agent

When SQL Server Agent jobs are not functioning as expected, security issues are a common cause of problems. To troubleshoot SQL Server Agent jobs, you should follow these steps:



1. Make sure that the job is running. Look in the Job Activity log and check to see when the job has run. For each failure of a job that is indicated by a red X, expand the job and find the job steps that are failing. The failing job steps will also have a red X icon—an example is shown on the slide.
2. Check the security account. When you click the job step that is failing, at the lower-right side of the window, there is an indication of the security context under which the job step ran. Check the group membership for the account to make sure that the account should have all required permissions.
3. Check the job owner. If a job is created on one instance of SQL Server Agent, and then deployed to another instance—for example, a development instance and a production instance—the job owner might exist in both instances, but the permissions of the job owner might be different. Make sure that the job owner has the correct login.
4. Check the securable objects that each job step task accesses. This includes any Transact-SQL objects that need to be accessed and any Windows files (including network paths) or other resources that need to be accessed.
5. For each failing step, check that the account that is being used to execute the step can access the resources that you have determined as necessary for the step.

Demonstration: Assigning a Security Context to Job Steps

In this demonstration, you will see the effect of SQL Server Agent job ownership on the security context of job steps.

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. In the **D:\Demofiles\Mod09** folder, right-click **Setup.cmd**, and then click **Run as Administrator**.
3. In the **User Account Control** dialog box, click **Yes**.

4. Start Microsoft SQL Server Management Studio, and connect to the **MIA-SQL** Database Engine instance by using Windows authentication.
5. On the **File** menu, point to **Open**, and then click **Project/Solution**.
6. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod09\Demo** folder.
7. Double-click the **Demo 1 - security context.sql** script file.
8. In Object Explorer, expand **SQL Server Agent**, expand **Jobs**, right-click **Record Execution Identity**, and then click **Start Job at Step**.
9. In the **Start Jobs - MIA-SQL** dialog box, make sure that the job ran successfully, and then click **Close**.

   The job triggers the **dbo.RecordIdentity** stored procedure in the **AdminDB** database. The procedure logs the identity of whoever ran the procedure to **dbo.IdentityLog**.
10. In Object Explorer, right-click **Record Execution Identity**, and then click **View History**.
11. In the **Log File Viewer - MIA-SQL** window, expand the visible job execution by clicking the plus sign on the row in the right pane, and then scroll the window to the right so that the **Message** column is visible.

    Notice that the first row shows that the job was invoked by **ADVENTUREWORKS\Student**, but the job step row shows that it was executed as **ADVENTUREWORKS\ServiceAcct**. When a **sysadmin** user owns a job, the job steps are executed in the context of the SQL Server Agent service account by default.

    Close the Log File Viewer - MIA-SQL window.
12. In the **Demo 1 - security context.sql** query pane, execute the query under the comment that begins **Task 1** to view the contents of the **AdminDB.dbo.IdentityLog** table. Notice that the identity of the service account was recorded.
13. In Object Explorer, right-click **Record Execution Identity**, and then click **Properties**.
14. In the **Job Properties - Record Execution Identity** window, on the **General** page, clear the **Owner** box, type **ITSupportLogin**, and then click **OK**.
15. In Object Explorer, right-click **Record Execution Identity**, and then click **Start Job at Step**.
16. In the **ITSupportLogin** dialog box, notice that the job fails, and then click **Close**.
17. To view the job history to find the reason for the failure, right-click **Record Execution Identity**, and then click **View History**.
18. In the **Log File Viewer - MIA-SQL** window, expand the failed job by clicking the plus sign next to the error symbol, scroll the window to the right until the **Message** column is visible, and then note the reason for the failure. The failure reason should show as follows:

    ```
    Executed as user: ITSupportLogin. The EXECUTE permission was denied on the object
    'RecordIdentity', database 'AdminDB', schema 'dbo'. [SQLSTATE 42000] (Error 229).
    The step failed.
    ```

    Close the Log File Viewer - MIA-SQL window.

    The job was run as the **ITSupportLogin** login, which maps to the **ITSupport** user in the **AdminDB** database; that user has no permissions to execute the stored procedure, so the job step failed.
19. In the **Demo 1 - security context.sql** query pane, execute the query under the comment that begins **Task 2** to grant the permission that is necessary to enable the job to run.
20. In Object Explorer, right-click **Record Execution Identity**, and then click **Start Job at Step**.

21. In the **Start Jobs - MIA-SQL** dialog box, notice that the job succeeds, and then click **Close**.
22. In the **Demo 1 - security context.sql** query pane, execute the query under the comment that begins **Task 1** to view the contents of the **AdminDB.dbo.IdentityLog** table.
23. On the **File** menu, click **Close**.
24. Keep SQL Server Management Studio open for the next demonstration.

Sequencing Activity

Put the following fixed roles in order from least privileged to most privileged by numbering each to indicate the correct order.

	Steps
	SQLAgentUserRole
	SQLAgentReaderRole
	SQLAgentOperatorRole
	sysadmin

Lesson 2
Configuring Credentials

For SQL Server Agent job steps to be able to access resources outside SQL Server, the job steps must be executed in the security context of a Windows identity that has permission to access the required resources. Windows identities are separate from SQL Server identities, even though SQL Server can utilize Windows logins and groups. For a job step to be able to use a separate Windows identity, the job step must be configured to log on as that identity. To be able to log on, the Windows user name and password need to be stored. Credentials are SQL Server objects that are used to store Windows user names and passwords.

Note: Credential objects can have other uses outside security in SQL Server Agent; for example, allowing logins that use SQL Server Authentication to impersonate a domain account.

For more information about credentials, see the topic *Credentials (Database Engine)* in Microsoft Docs:

Credentials (Database Engine)

http://aka.ms/J6i9jw

Lesson Objectives

After completing this lesson, you will be able to:

- Describe credentials
- Configure credentials
- Manage credentials

Overview of Credentials

A credential is a SQL Server object that contains the authentication information that is required to connect to a resource outside SQL Server. Most credentials have a Windows user name and password, although they might contain authentication information for a third-party cryptographic provider.

- Authentication for a resource or system outside the database engine instance
  - Typically Windows user name and password
  - Third-party cryptographic providers are also supported
- Some system credentials are created automatically during SQL Server installation (## prefix)

Credentials

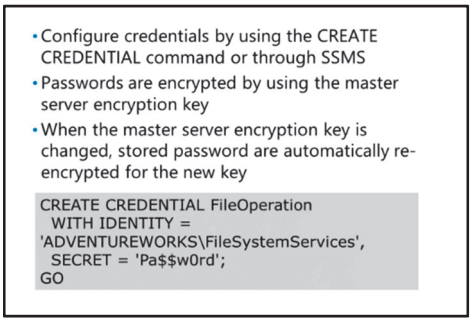
A SQL Server Agent proxy account that can be used for job execution maps to a credential in SQL Server. In the next lesson, you will see how to map a proxy account to a credential.

SQL Server automatically creates some system credentials that are associated with specific endpoints. These automatically created system credentials have names that are prefixed with two hash signs (##).

Proxy accounts in SQL Server Agent cannot use system credentials; proxy accounts require server-level credentials that users define.

Configuring Credentials

Credentials can be created by using the Transact-SQL CREATE CREDENTIAL statement, or by using SQL Server Management Studio.



Configuring Credentials

The password for a credential is called a secret and is strongly encrypted and stored in the **master** database. When SQL Server first needs to perform any type of encryption, the SQL Server Agent service generates a service master encryption key. The service master key is also used to protect the master keys for each database. (Not all databases have master keys.)

Often an organization will have a policy that requires encryption keys to be replaced on a regular basis. If the service master key is regenerated, the secrets that are stored for credentials are automatically decrypted and re-encrypted by using the new service master key.

 **Note:** Encryption in SQL Server is an advanced topic that is outside the scope of this course. Note also that the encryption of secrets for credentials by using an Extensible Key Management (EKM) provider is supported, but is also beyond the scope of this course.

The following example creates the **FileOperation** credential for the Windows account **ADVENTUREWORKS\FileSystemServices** with the password **Pa55w.rd**:

CREATE CREDENTIAL

```
CREATE CREDENTIAL FileOperation
  WITH IDENTITY = 'ADVENTUREWORKS\FileSystemServices',
  SECRET = 'Pa55w.rd';
GO
```

For more information about creating credentials, see the topic *Create a Credential* in Microsoft Docs:

 **Create a Credential**

http://aka.ms/H7v4s5

For more information about working with encryption keys in SQL Server, see the topic *SQL Server and Database Encryption Keys (Database Engine)* in Microsoft Docs:

 **SQL Server and Database Encryption Keys (Database Engine)**

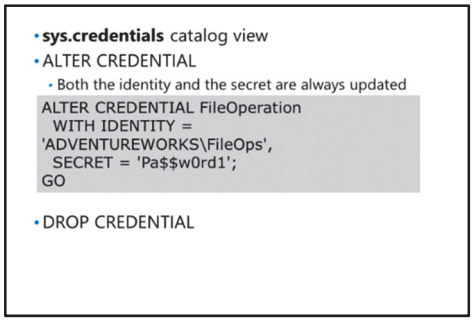
http://aka.ms/Lmw2y6

Managing Credentials

The **sys.credentials** system view provides catalog information about existing credentials. For more information about **sys.credentials**, see the topic *sys.credentials (Transact-SQL)* in Microsoft Docs:

sys.credentials (Transact-SQL)

http://aka.ms/Cr8uot



Modifying Credentials

The password for a Windows account could change over time. You can update a credential with new values by using the ALTER CREDENTIAL statement. In the example on the slide, notice how the **FileOperation** credential is being updated. Both the user name and password (that is, the secret) are supplied in the ALTER CREDENTIAL statement. Although the secret is optional, the ALTER CREDENTIAL command always updates both the identity and the secret. If the secret is not supplied to an ALTER CREDENTIAL statement, the secret is set to NULL.

Credentials are removed by the DROP CREDENTIAL statement.

For more information about managing credentials by using ALTER CREDENTIAL, see the topic *ALTER CREDENTIAL (Transact-SQL)* in Microsoft Docs:

ALTER CREDENTIAL (Transact-SQL)

http://aka.ms/Ci5lfx

Demonstration: Configuring Credentials

In this demonstration, you will see how to work with credentials.

Demonstration Steps

1. In SQL Server Management Studio, in Object Explorer, under **MIA-SQL**, under **SQL Server Agent**, right-click **Jobs**, and then click **New Job**.
2. In the **New Job** window, on the **General** page, in the **Name** box, type **Copy Export File**.
3. On the **Steps** page, click **New**.
4. In the **New Job Step** window, on the **General** page, in the **Step name** box, type **Copy File**.
5. In the **Type** list, click **Operating System (CmdExec)**.
6. In the **Command** window, type the following:

```
copy d:\demofiles\Mod09\ExportData\export.txt
d:\demofiles\Mod09\ImportData\import.txt /Y
```

   Notice that the job is configured to run in the security context of the SQL Server Agent service account.

7. On the **Advanced** page, in the **On success action** box, click **Quit the job reporting success**, and then click **OK**.
8. In the **New Job** window, click **OK** to create the job.

9. In Object Explorer, under **Jobs**, right-click **Copy Export File**, and then click **Start Job at Step**.
10. In the **Start Jobs - MIA-SQL** dialog box, notice that the job fails, and then click **Close**.
11. To view the job history to find the reason for the failure, right-click **Copy Export File**, and then click **View History**.
12. In the **Log File Viewer - MIA-SQL** window, expand the failed job by clicking the plus sign next to the error symbol, scroll the window to the right until the **Message** column is visible, and then note the reason for the failure. The failure reason should show as follows:

```
Executed as user: ADVENTUREWORKS\ServiceAcct. Access is denied.  Process Exit Code 1.
The step failed.
```

    Close the Log File Viewer - MIA-SQL window. The job step failed because the service account does not have permission to access the source and target folders in the file system. (One solution would be to grant access to the folders to the service account, but instead you will create a credential, and then link it to a proxy account in the next demonstration.)

13. In Solution Explorer, double-click the **Demo 2 - credential.sql** script file.
14. Execute the code under the comment that begins **Task 1** to create a credential called **FileOperation** that is linked to the **ADVENTUREWORKS\FileOps** domain user with the secret **Pa55w.rd**.
15. Execute the code under the comment that begins **Task 2** to examine the contents of the **sys.credentials** catalog view. One row should be returned for the credential that you have just created.
16. On the **File** menu, click **Close**.
17. Leave SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
What happens to a credential when the password of the Windows user that the credential references is changed?	
Select the correct answer.	
	The credential is automatically deleted.
	The credential is disabled.
	Attempts to use the credential fail until the password is updated.
	The credential continues to operate normally.

Lesson 3
Configuring Proxy Accounts

In the last lesson, you saw that credentials are used in SQL Server to store identities that are external to SQL Server—typically Windows user names and passwords. To enable a job step in a SQL Server Agent job to use a credential, the job step is configured to use the security context of the credential through a proxy account.

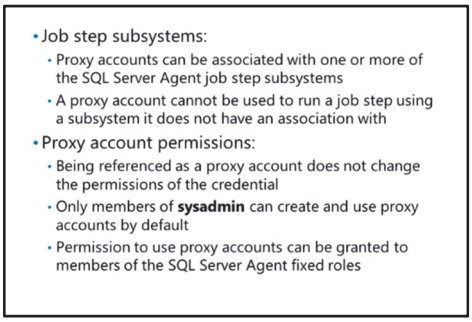
Lesson Objectives

After completing this lesson, you will be able to:

- Describe proxy accounts
- Manage proxy accounts

Overview of Proxy Accounts

A proxy account in SQL Server Agent is used to define the security context that is used for a job step. A proxy account is typically used to provide SQL Server Agent with access to the security credentials for a Windows user.



Note: When members of **sysadmin** create job steps that run in the security context of the SQL Server Agent service account, they make use of a built-in proxy account for the service account. Permission to use this built-in proxy account cannot be granted to users who do not belong to **sysadmin**.

Job Step Subsystems

Each of the job step types that are not based on Transact-SQL is referred to as a subsystem. Subsystems assist in providing security control because they segment the functions that are available to a proxy account. Each proxy account can be associated with one or more subsystems. The list of available subsystems is as follows:

- ActiveX® Script
- Operating System (CmdExec)
- Windows PowerShell®
- Replication Distributor
- Replication Merge
- Replication Queue Reader
- Replication Snapshot
- Replication Transaction-Log Reader
- SQL Server Analysis Services Command

- SQL Server Analysis Services Query
- SQL Server Integration Services Package

Note: The ActiveX Script subsystem is marked for deprecation in a future version of SQL Server, and should not be used for new development.

A job step that uses the proxy account can access the specified subsystems by using the security context of the Windows user. Before SQL Server Agent runs a job step that uses a proxy account, SQL Server Agent impersonates the credentials that are defined in the proxy account, and then runs the job step by using that security context.

Note: The Windows user who is specified in the credential must have the **Log on as a batch job** permission on the computer on which SQL Server is running.

Proxy Account Permissions

The creation of a proxy account does not change existing permissions for the Windows account that is specified in the credential. For example, you can create a proxy account for a Windows account that does not have permission to connect to an instance of SQL Server. Job steps that use that proxy account would be unable to connect to SQL Server when executing commands.

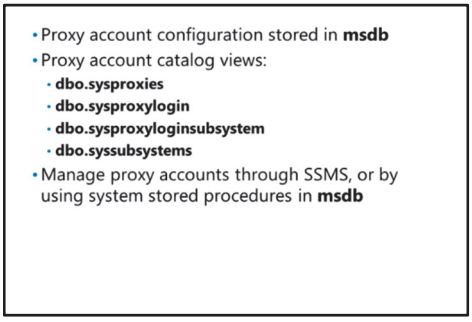
Note that a user must have permission to use a proxy account before they can specify the proxy account in a job step. By default, only members of the **sysadmin** fixed server role have permission to access all proxy accounts. Permission to use individual proxy accounts can be granted to members of the SQL Server Agent fixed roles—**SQLAgentUserRole**, **SQLAgentReaderRole**, and **SQLAgentOperatorRole**.

Note: As with other permissions in SQL Server Agent, access to proxy accounts is inherited from lower-privileged SQL Server Agent fixed roles to higher-privileged SQL Server Agent fixed roles. For example, if **SQLAgentUserRole** is granted permission to use the **FileOpProxy** proxy account, members of both **SQLAgentReaderRole** and **SQLAgentOperatorRole** will also have permission to use the proxy account.

Managing Proxy Accounts

In common with other elements of SQL Server Agent configuration, proxy accounts are stored in the **msdb** system database.

A set of system catalog views that describe proxy account configuration is available.



View name	Description
dbo.sysproxies	Returns one row per proxy account.
dbo.sysproxylogin	Returns details of which security principals have permission to work with each proxy account. Note that no entry for members of the **sysadmin** role is stored or returned.
dbo.sysproxyloginsubsystem	Returns which SQL Server Agent subsystems are associated with each proxy account.
dbo.syssubsystems	Returns information about all available SQL Server Agent subsystems.

Proxy accounts can be configured by using SQL Server Management Studio, or through system stored procedures in the **msdb** system database.

For more information about creating proxy accounts, see the topic *Create a SQL Server Agent Proxy* in Microsoft Docs:

Create a SQL Server Agent Proxy

http://aka.ms/Ka0f55

For more information about modifying proxy accounts, see the topic *Modify a SQL Server Agent Proxy* in Microsoft Docs:

Modify a SQL Server Agent Proxy

http://aka.ms/E3sbe1

Demonstration: Configuring Proxy Accounts

In this demonstration, you will see how to configure and use a SQL Server Agent proxy account.

Demonstration Steps

1. In SQL Server Management Studio, in Solution Explorer, double-click the **Demo 3 - proxy.sql** script file.
2. Execute the code under the comment that begins **Task 1** to create a new proxy account that is linked to the **FileOperation** credential that you created in the last demonstration.
3. Execute the code under the comment that begins **Task 2** to examine the **dbo.sysproxies** catalog view.
4. Execute the code under the comment that begins **Task 3** to examine the contents of the **dbo.syssubsystems** system view. Note that the **CmdExec** subsystem has a **subsystem_id** of **3**.
5. Execute the code under the comment that begins **Task 4** to associate the **FileOp** proxy account with the **CmdExec** subsystem (**@subsystem_id = 3**).
6. In Object Explorer, under **Jobs**, right-click **Copy Export File**, and then click **Properties**.
7. In the **Job Properties - Copy Export File** window, on the **Steps** page, click **Edit**.
8. In the **Job Step Properties - Copy File** window, in the **Type** list, click **Operating system (CmdExec)**.

9. In the **Run as** list, click **FileOp**, and then click **OK**.
10. In the **Job Properties - Copy Export File** window, click **OK**.
11. In File Explorer, browse to **D:\Demofiles\Mod09\ImportData** to demonstrate that the folder is empty.
12. In SQL Server Management Studio, in Object Explorer, under **Jobs**, right-click **Copy Export File**, and then click **Start Job at Step**.
13. In the **Start Jobs - MIA-SQL** dialog box, notice that the job succeeds, and then click **Close**.
14. In File Explorer, demonstrate that the folder now contains a copy of the file from the **ExportData** folder, and then close File Explorer.
15. Close SQL Server Management Studio without saving any changes.

Question: Why are credentials stored in the **master** system database and proxy accounts stored in the **msdb** system database?

Lab: Configuring Security for SQL Server Agent

Scenario

A new SQL Server Integration Services package that is triggered by a SQL Server Agent job on the MIA-SQL instance is failing. It is likely that a security issue is causing the job to fail. In this lab, you will review the reasons why the job is failing, correct the situation through the use of credentials and proxy accounts, and assign a proxy account to the SQL Server Agent job to correct the issue.

Objectives

After completing this lab, you will be able to:

- Analyze security problems in SQL Server Agent.
- Configure credentials.
- Configure proxy accounts.
- Customize the security context of SQL Server Agent job steps.
- Test the security context of SQL Server Agent jobs and job steps.

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Analyzing Security Problems in SQL Server Agent

Scenario

The developer of the SQL Server Integration Services package informs you that the purpose of the package is to generate a report from the **InternetSales** database and export the results to a file called **sales_log.csv** in **D:\Labfiles\Lab09\Starter\SalesLog** on the **20764C-MIA-SQL** machine. The SQL Server Agent job that triggers the job is called **Generate Sales Log**, and is scheduled to execute every 10 minutes. However, the expected file is not being generated, so the developer believes that the job is failing.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Examine the Job History for the Failing Job

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Examine the Job History for the Failing Job

1. Use SQL Server Management Studio to connect to the **MIA-SQL** Database Engine instance, and then examine the job history for the failing **Generate Sales Log** job.
2. Identify the owner of the **Generate Sales Log** job.

3. Based on the information in the job history, what is the cause of the job failure?

Results: After completing this exercise, you should have identified the cause of the job failure.

Exercise 2: Configuring a Credential

Scenario

You have identified that the job is failing because the job owner is not a member of a role that allows the execution of a SQL Server Integration Services package without a proxy account. After discussion with the developer and the domain administrator, you decide to create a proxy account that is suitable for use with this task.

Before you can create a proxy account, you must configure a credential.

The main tasks for this exercise are as follows:

1. Create a Credential

▶ Task 1: Create a Credential

- Use SQL Server Management Studio to add a new credential called **ExtractUser** that references the **ADVENTUREWORKS\Student** domain account. The password for the account is **Pa55w.rd**.

  Hint: Credentials are configured under the server-level **Security** node in Object Explorer.

Results: After completing this exercise, you should have created a credential that references the **ADVENTUREWORKS\Student** Windows account.

Exercise 3: Configuring a Proxy Account

Scenario

You need to create a proxy account by using the credential that you created in the last exercise, configure the proxy account for access to the SQL Server Integration Services subsystem, and grant permission to use the proxy account to the owner of the SQL Server Agent job called **Generate Sales Log**.

The main tasks for this exercise are as follows:

1. Create a Proxy Account

▶ Task 1: Create a Proxy Account

1. Using SQL Server Management Studio, create a proxy account called **ExtractProxy** that uses the security context of the **ExtractUser** credential that you created in the last exercise.
2. Configure **ExtractProxy** to have access to the **SQL Server Integration Services Package** job step subsystem.
3. Grant permission to use **ExtractProxy** to the owner of the SQL Server Agent job called **Generate Sales Log**.

Results: After completing this exercise, you should have created a proxy account that is suitable for correcting the problem with the SQL Server Agent job called **Generate Sales Log**.

Exercise 4: Configuring and Testing the Security Context of the Job

Scenario

Using the proxy account that you created in the last exercise, you will amend the configuration of the **Generate Sales Log** SQL Server Agent job to use the proxy account when executing the SQL Server Integration Services package. After the job configuration is updated, you will test the job and verify that it works as expected.

The main tasks for this exercise are as follows:

1. Configure the Job to Use a Proxy Account

2. Test the Configuration

▶ Task 1: Configure the Job to Use a Proxy Account

- Edit the configuration of the **Generate Sales Log** SQL Server Agent job to use **ExtractProxy** as the security context for the **Execute Package** job step.

▶ Task 2: Test the Configuration

1. Execute the **Generate Sales Log** SQL Server Agent job and verify that the job runs correctly.
2. Check that **sales_log.csv** has been generated in **D:\Labfiles\Lab09\Starter\SalesLog**.
3. Close any open applications without saving changes.

Results: After completing this exercise, the **Generate Sales Log** SQL Server Agent job should be working correctly, and the **sales_log.csv** file should be generated to **D:\Labfiles\Lab09\Starter\SalesLog** each time the job runs.

Check Your Knowledge

Question	
The SQL Server Integration Services package in this lab uses SQL Server Authentication to connect to the MIA-SQL instance to extract data. If the SQL Server Integration Services package were configured to use Windows authentication for its database connection, under what security context is the connection made when the Generate Sales Log job is executed by ADVENTUREWORKS\Administrator? **Assume that the exercise was successfully completed when you are selecting your answer.**	
Select the correct answer.	
	ADVENTUREWORKS\Administrator
	PromoteApp
	ADVENTUREWORKS\Student
	The SQL Server Agent service account

Module Review and Takeaways

In this module, you have learned about configuring security for the SQL Server Agent service and for SQL Server Agent job steps.

Best Practice:

1. Use a Windows domain user as the SQL Server Agent service account.
2. Use an account that has least privileges.

Create proxy accounts that have least permissions assigned for job execution.

Review Question(s)

Question: As a general rule, why should proxy accounts not be assigned access to all of the job step subsystems?

Module 10

Monitoring SQL Server with Alerts and Notifications

Contents:

Module Overview

One key aspect of managing Microsoft SQL Server® in a proactive manner is to make sure you are aware of problems and events that occur in the server, as they happen. SQL Server logs a wealth of information about issues. You can configure it to advise you automatically when these issues occur, by using alerts and notifications. The most common way that SQL Server database administrators receive details of events of interest is by email message. This module covers the configuration of Database Mail, alerts, and notifications for a SQL Server instance, and the configuration of alerts for Microsoft Azure SQL Database.

Objectives

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

- Monitor SQL Server errors.
- Configure database mail.
- Configure operators, alerts, and notifications.
- Work with alerts in Azure SQL Database.

Lesson 1
Monitoring SQL Server Errors

It is important to understand the core aspects of errors as they apply to SQL Server. In particular, you need to consider the nature and locations of errors, in addition to the data that they return. SQL Server records severe errors in the SQL Server error log, so it is also important to know how to configure the log.

Lesson Objectives

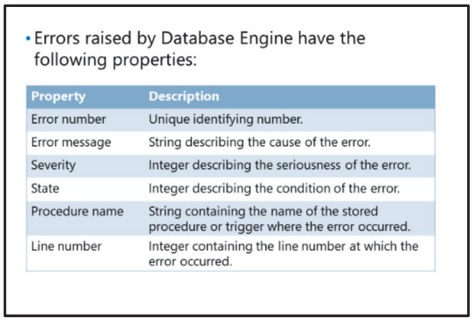
After completing this lesson, you will be able to:

- Define SQL Server errors.
- Describe error severity levels.
- Configure the SQL Server error log.

What Is an Error?

It might not be immediately obvious that a SQL Server Database Engine error (or exception) is itself an object, and therefore has properties that you can access.

Property	Description
Error number	Unique identifying number.
Error message	String describing the cause of the error.
Severity	Int describing the seriousness of the error.
State	Int describing the condition of the error.
Procedure name	String containing the name of the stored procedure or trigger where the error occurred.
Line number	Int containing the line number at which the error occurred.



Error numbers are helpful when trying to locate information about the specific error, particularly when searching for information online.

The following example shows how to use the **sys.messages** catalog view to retrieve a list of system supplied error messages, showing the error number (**message_id**), severity (**severity**), and error message (**text**) properties described in the table above in English:

Viewing System Error Messages

```
SELECT * FROM sys.messages WHERE language_id = 1033
ORDER BY message_id;
```

Error messages can be localized and are returned in a range of languages, so the WHERE clause of this example limits the results to view only the English version.

Note: The **State** property is not defined in **sys.messages**. A given error message will always have the same severity, but might occur with different states.

Error numbers less than 50,000 are reserved for system error messages. You can define custom error messages that are specific to your applications, using error numbers greater than 50,000.

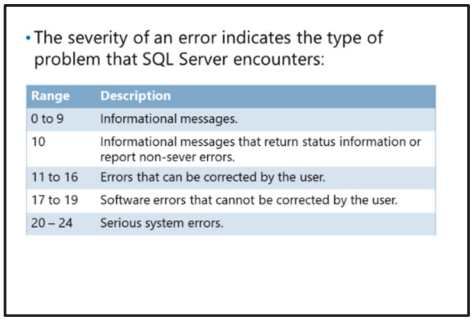
For more information about **sys.messages** and Database Engine errors, see the topic *Understanding Database Engine Errors* in the SQL Server Technical Documentation:

Understanding Database Engine Errors

https://aka.ms/M6lfd2

Error Severity Levels

The severity of an error indicates the type of problem encountered by the Database Engine. Low severity values are informational messages and do not indicate true errors. Error severities are grouped into ranges.



Severities 0 to 10

Severity values from 0 to 10 are informational messages raised by the Database Engine to provide information associated with the running of a query.

For example, consider the query shown below.
This query returns a count, but on the **Messages** tab in SQL Server Management Studio (SSMS), the message "Warning: Null value is eliminated by an aggregate or other SET operation" is also displayed. No error has occurred, but SQL Server warns that it ignored NULL values when counting the rows:

Information Messages

```
SELECT COUNT(Color) FROM Production.Product
```

Messages with a severity of 10 or lower are treated as information, and are ignored by the Transact-SQL TRY…CATCH construct. An error raised with severity 10 is converted to severity 0 before being returned to a client application.

Severities 11 to 16

Severity values from 11 to 16 are used for errors that the user can correct. Typically, SQL Server uses them when it asserts that the code being executed contains an error, or has experienced an error during execution. Errors in this range include:

- **11**. Indicates that an object does not exist.
- **13**. Indicates a transaction deadlock.
- **14**. Indicates errors such as permission denied.
- **15**. Indicates syntax errors in a Transact-SQL command.

When an error with severity in the range 11 to 16 is raised, the Transact-SQL batch where the error occurred can continue to execute.

Severities 17 to 19

Severity values 17 to 19 are serious software errors that the user cannot correct, but which can normally be addressed by an administrator. For example, severity 17 indicates that SQL Server has run out of resources (such as memory or disk space). When an error with severity in the range 17 to 19 is raised, the Transact-SQL batch where the error occurred will typically abort, but the database connection is not affected. Errors with severity 19 are written to the SQL Server error log.

Severities 20 to 24

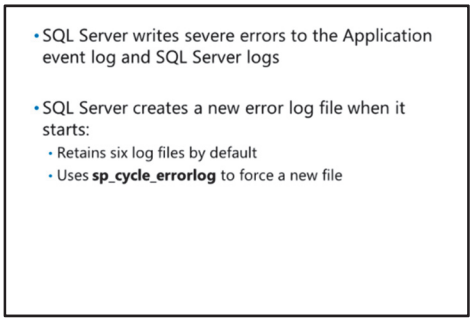
Severity values of 20 and above indicate very serious errors that involve either the hardware or SQL Server itself, which threatens the integrity of one or more databases or the SQL Server service. Errors in the severity range 20 to 24 are written to the SQL Server error log. The connection on which a severity error of 20 or greater occurs will, in almost all circumstances, be automatically closed.

For more information about Database Engine error severity levels, see the topic *Database Engine Error Severities* in the SQL Server Technical Documentation:

Database Engine Error Severities

https://aka.ms/A4jqp2

Configuring the SQL Server Error Log



Important messages, particularly those considered as severe error messages, are logged to both the Windows application event log and SQL Server error log. The **sys.messages** view shows the available error messages and indicates which ones will be logged by default—rows in **sys.messages** with an **is_event_logged** value of 1.

SQL Server error log files are text files that you can view by using any text editor, or the Log Viewer provided by SSMS. By default, SQL Server writes its error logs to the ERRORLOG folder in the servers specific instance within the Program Files folder. The log file location is passed to the SQL Server service as a startup parameter; you can customize it by changing the service startup parameter, either when SQL Server is installed, or by changing the parameter through SQL Server Configuration Manager.

By default, SQL Server retains backups of the previous six logs and gives the most recent log backup the extension .1, the second most recent the extension .2, and so on. The current error log has no extension. You can increase the number of log files to retain by customizing the log configuration, but you cannot choose to retain fewer than six log files.

The log file cycles with every restart of the SQL Server instance. Occasionally, you might want to remove excessively large log files. You can use the **sp_cycle_errorlog** system stored procedure to close the existing log file and open a new one on demand. If there is a regular need to recycle the log file, you could create a SQL Server Agent job to execute the system stored procedure on a schedule. Cycling the log can help you to stop the current error log becoming too large.

For more information about **sp_cycle_errorlog**, see the topic *sp_cycle_errorlog (Transact-SQL)* in the SQL Server Technical Documentation:

sp_cycle_errorlog (Transact-SQL)

https://aka.ms/Lz9o3t

For more information about increasing the number of log files retained before recycling through SSMS, see the topic *Configure SQL Server Error Logs* in the SQL Server Technical Documentation:

Configure SQL Server Error Logs

https://aka.ms/Uk4gfi

Demonstration: Working with the Database Engine Error Log

In this demonstration, you will see how to:

- View the SQL Server error log.
- Cycle the log file.

Demonstration Steps

View the SQL Server Error Log

1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and **20764C-MIA-SQL** virtual machines are running, and log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55.wrd**.
2. In the **D:\Demofiles\Mod10** folder, run **Setup.cmd** as Administrator.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** Database Engine instance using Windows authentication.
4. In Object Explorer, under **MIA-SQL**, expand **Management**, and then expand **SQL Server Logs**.
5. Right-click **Current**, and then click **View SQL Server Log**.
6. In the **Log File Viewer - MIA-SQL** window, view the log entries. Note that when you select a log entry, its details are shown in the lower pane.
7. In the **Select logs** pane, expand **SQL Server Agent**, and select **Current**.
8. Scroll the main log entries pane to the right until you can see the **Log Type** column, and then scroll down to find an entry with the log type **SQL Server Agent**.
9. When you have finished viewing the log entries, click **Close**.

10. In File Explorer, navigate to the **C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Log** folder.
11. If the **Log** dialog box appears, click **Continue**.
12. If the **User Account Control** dialog box appears, click **Yes**.
13. Note that the current SQL Server log is stored here in the file named **ERRORLOG**, and the current SQL Server Agent log is stored as **SQLAGENT.1**. The remaining log files contain log entries for other SQL Server components and services.

Cycle the Log File

1. In SSMS, click **New Query**.
2. In the query window, type the following Transact-SQL code, and then click **Execute**:

```
EXEC sys.sp_cycle_errorlog;
```

3. In Object Explorer, under SQL Server Logs, right-click **Current**, and then click **View SQL Server Log**.
4. In the **Log File Viewer - MIA-SQL** window, note that the log has been reinitialized, and then click **Close**.
5. Close the query window without saving changes, but leave SSMS open for the next demonstration.

Check Your Knowledge

Question	
If an error message is for information only, which of the following ranges will its severity fall into?	
Select the correct answer.	
	0 to 10
	11 to 16
	17 to 19
	20 to 24

Lesson 2
Configuring Database Mail

SQL Server can be configured to advise administrators when issues arise that require their attention, such as the failure of a scheduled job or a significant error. Email is the most commonly-used mechanism for notifications from SQL Server. You can use the Database Mail feature of SQL Server to connect to an existing Simple Mail Transport Protocol (SMTP) server when SQL Server needs to send email.

Database Mail is not solely used for sending alerts and notifications to administrators; when Database Mail is configured, you can send email from Transact-SQL code. To restrict this permission, you can control which users can utilize the email features of the product, and configure different email profiles for use by different security principals. Because it can be important to be able to track and trace sent emails, SQL Server enables you to configure a policy for their retention.

Note: Database Mail is not available in Azure SQL Database—this uses a different alerting mechanism, discussed later in this module.

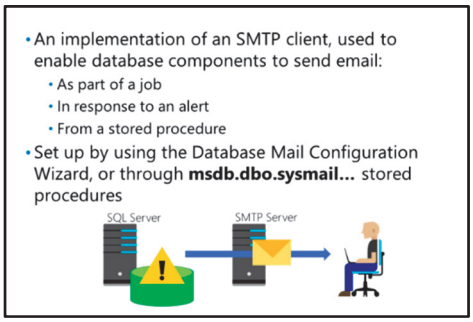
Lesson Objectives

After completing this lesson, you will be able to:

- Describe Database Mail.
- Configure Database Mail profiles.
- Configure Database Mail security.
- Configure Database Mail retention.

Overview of Database Mail

Database Mail sends email by using an SMTP server. It is designed to be a reliable, scalable, secure, and supportable system. Messages are delivered asynchronously, by a process outside the Database Engine, to avoid impacting the performance of your database system. If the SMTP server is unavailable, SQL Server can queue the messages until the service is available again. You can use Database Mail to send email messages as part of a SQL Server Agent job, in response to an alert, or from Transact-SQL using the **sp_send_dbmail** system stored procedure.



By default, the Database Mail stored procedures are disabled to reduce the surface area of SQL Server; when they are enabled, only users who are members of the **sysadmin** server role or the **DatabaseMailUserRole** database role in the **msdb** database can execute them. Database Mail logs email activity, and also stores copies of all messages and attachments in the **msdb** database.

You use the Database Mail Configuration Wizard to enable Database Mail and to configure accounts and profiles. A Database Mail account contains all the information that SQL Server needs to send an email message to the mail server. You must specify what type of authentication to use (Windows, basic, or anonymous), the email address, the email server name, type, and port number—and if your SMTP server

requires authentication, a username and password. You can also configure Database Mail using the group of system stored procedures with the naming pattern **msdb.dbo.sysmail...**.

SQL Server stores the configuration details in the **msdb** database, along with all SQL Server Agent configuration data. SQL Server Agent also caches the profile information in memory so you can send email if the SQL Server Database Engine is no longer available.

For a further overview of Database Mail, see the topic *Database Mail* in the SQL Server Technical Documentation:

Database Mail

https://aka.ms/Gjok0m

For more information about the process that sends Database Mail, see the topic *Database Mail External Program* in the SQL Server Technical Documentation:

Database Mail External Program

https://aka.ms/In5xdu

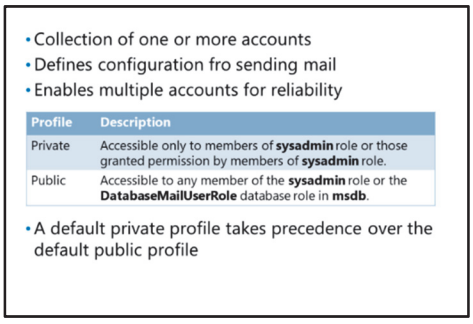
For a complete list of the stored procedures available to configure and work with Database Mail, see the topic *Database Mail Stored Procedures (Transact-SQL)* in the SQL Server Technical Documentation:

Database Mail Stored Procedures (Transact-SQL)

https://aka.ms/Ip6cb0

Database Mail Profiles

A Database Mail profile is a collection of one or more Database Mail accounts. When there is more than one account in a profile, Database Mail tries to send email using the accounts in a predefined order, ensuring that, if one email server is unresponsive, another can be used.



Profiles can be private or public. Private profiles are strictly controlled and are only available to members of the **sysadmin** role or those granted permission to the profile by members of the **sysadmin** role. In contrast, any database principal that is a member of the **DatabaseMailUserRole** in **msdb** can use a public profile.

Mail Profiles

You can create multiple configurations by using different profiles. For example, you could create one profile to send mail to an internal SMTP server—using an internal email address—for mails sent by SQL Server Agent, and a second profile for a database application to send external email notifications to customers or suppliers.

Each database user can access multiple profiles. If you do not specify a profile when sending an email message, Database Mail uses the default profile. If both private and public profiles exist, precedence is given to a private default profile over a public one. If you do not specify a default profile, or if a

nondefault profile should be used, you must specify the profile name you want to use as a parameter when sending mail.

The following example shows how to use the **sp_send_dbmail** system stored procedure to send an email message using a specific profile, and using the optional **@profile_name** parameter:

Specifying a Database Mail Profile

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'HR Administrator',
    @recipients = 'admin@AdventureWorks.com',
    @body = 'Daily backup completed successfully.',
    @subject = 'Daily backup status';
```

For more information about profiles in Database Mail, see the topic *Database Mail Configuration Objects* in the SQL Server Technical Documentation:

Database Mail Configuration Objects

https://aka.ms/R7s4b4

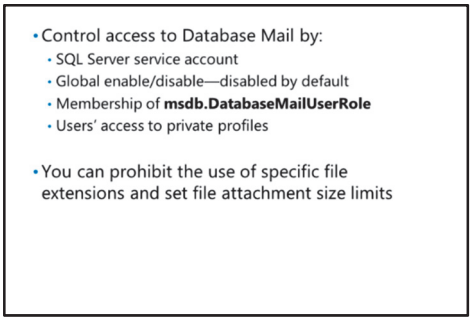
The **msdb.dbo.sp_send_dbmail** stored procedure has a lot of powerful features, including the ability to send email messages in HTML or plain text format, and the ability to attach files to email messages. For more information about the capabilities of **sp_send_dbmail**, see the topic *sp_send_dbmail (Transact-SQL)* in the SQL Server Technical Documentation:

sp_send_dbmail (Transact-SQL)

https://aka.ms/H9ejv2

Database Mail Security

It is important to consider the service account that the SQL Server service will use when you are configuring Database Mail. If you configure SQL Server to run as the Local Service account, it does not have permission to make nonanonymous outgoing network connections. In that case, Database Mail might not be able to connect to an SMTP server running on a different computer.



Database Mail Stored Procedures

To minimize the security surface of SQL Server, the system extended stored procedures for Database Mail are disabled by default. When you run the Database Mail Configuration Wizard, it automatically enables the procedures. To manually configure Database Mail, you can enable the Database Mail system extended stored procedures by using the **sp_configure** stored procedure, setting the **Database Mail XPs** option to the value of **1**.

Security and Attachment Limitations

Not all SQL Server users can send email messages. This is limited to members of the database role called **DatabaseMailUserRole** in the **msdb** database. Members of the **sysadmin** fixed server role can also send Database Mail.

You can limit the types and size of attachments that users can send in email messages by Database Mail. You can configure this limitation by using the Database Mail Configuration Wizard or by calling the **dbo.sysmail_configure_sp** system stored procedure in the **msdb** database.

For more information about **dbo.sysmail_configure_sp**, see the topic *sysmail_configure_sp (Transact-SQL)* in the SQL Server Technical Documentation:

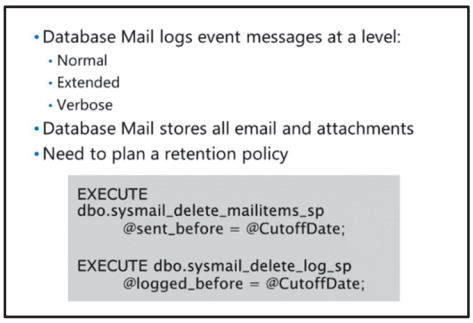
sysmail_configure_sp (Transact-SQL)

https://aka.ms/neysmz

Database Mail Logs and Retention

SQL Server logs event messages in internal tables in the **msdb** database. You can configure how much information it logs by setting the logging level to one of the following:

- **Normal**. Only logs errors.
- **Extended**. Logs errors, warnings, and information messages.
- **Verbose**. Logs extended messages, in addition to success messages and a number of internal messages.



Note: You should only use the verbose level for troubleshooting purposes, because it can generate a large volume of log entries that can cause the database to grow, and potentially impact performance.

You configure the logging level parameter by using the **Configure System Parameters** page of the Database Mail Configuration Wizard or by calling the **dbo.sysmail_configure_sp** stored procedure in the **msdb** database. You can view the logged messages by querying the **dbo.sysmail_event_log** table.

Internal tables in the **msdb** database also hold copies of the email messages and attachments that Database Mail sends, together with the current status of each message. Database Mail updates these tables when it processes messages. You can track the delivery status of an individual message by viewing information in the following system views in **msdb**:

- **dbo.sysmail_allitems**—shows all mail messages.
- **dbo.sysmail_sentitems**—shows sent mail messages.
- **dbo.sysmail_unsentitems**—shows mail messages that are queued to be sent.
- **dbo.sysmail_faileditems**—shows mail messages where sending has failed.

To see details of email attachments, query the **dbo.sysmail_mailattachments** view.

Because Database Mail retains the outgoing messages and their attachments, you need to plan a retention policy for this data. If the volume of Database Mail messages and related attachments is high, plan for substantial growth of the **msdb** database.

To restrict the growth of **msdb** due to retained Database Mail messages, you can periodically delete messages to regain space and to comply with your organization's document retention policies.

The following example shows how to delete messages, attachments, and log entries that are more than one month old:

Deleting Old Logs and Database Mail Items

```
USE msdb;
GO

DECLARE @CutoffDate datetime ;
SET @CutoffDate = DATEADD(m, -1,SYSDATETIME());

EXECUTE dbo.sysmail_delete_mailitems_sp
        @sent_before = @CutoffDate;

EXECUTE dbo.sysmail_delete_log_sp
        @logged_before = @CutoffDate;
```

You could schedule these commands to be executed periodically by creating a SQL Server Agent job.

For more information about the views and stored procedures available to work with Database Mail messaging objects, see the topic *Database Mail Messaging Objects* in the SQL Server Technical Documentation:

Database Mail Messaging Objects

https://aka.ms/Nha0um

Demonstration: Configuring Database Mail

In this demonstration, you will see how to:

- Create a Database Mail profile.
- Send a test email message.
- Query Database Mail system tables.

Demonstration Steps

Create a Database Mail Profile

1. In SSMS, in Object Explorer, under **MIA-SQL**, under **Management**, right-click **Database Mail**, and then click **Configure Database Mail**.
2. On the **Welcome to Database Mail Configuration Wizard** page, click **Next**.
3. On the **Select Configuration Task** page, click **Set up Database Mail by performing the following tasks:**, and then click **Next**.
4. On the **New Profile** page, in the **Profile name** box, type **SQL Server Agent Profile**, and then click **Add**.
5. In the **Add Account to Profile 'SQL Server Agent Profile'** dialog box, click **New Account**.
6. In the **New Database Mail Account** dialog box, enter the following details, and then click **OK**:
    - **Account name**: AdventureWorks Administrator
    - **E-mail address**: administrator@adventureworks.msft
    - **Display name**: Administrator (AdventureWorks)
    - **Reply e-mail**: administrator@adventureworks.msft

- **Server name**: mia-sql.adventureworks.msft

7. On the **New Profile** page, click **Next**.
8. On the **Manage Profile Security** page, select **Public** for the **SQL Server Agent Profile** profile, and set its **Default Profile** setting to **Yes**, and then click **Next**.
9. On the **Configure System Parameters** page, click **Next**.
10. On the **Complete the Wizard** page, click **Finish**, and when configuration is complete, click **Close**.

Send a Test Email Message

1. In Object Explorer, right-click **Database Mail**, and then click **Send Test E-Mail**.
2. In the **Send Test E-Mail from MIA-SQL** dialog box, ensure that the **SQL Server Agent Profile** database mail profile is selected.
3. In the **To** box, type **student@adventureworks.msft**, and then click **Send Test E-mail**.
4. In File Explorer, navigate to the **C:\inetpub\mailroot\Drop** folder, and verify that an email message has been created.
5. Double-click the message to view it in Outlook. When you have read the message, close it and minimize the **Drop** folder window.
6. In the **Database Mail Test E-Mail** dialog box (which might be behind SQL Server Management Studio), click **OK**.

Query Database Mail System Tables

1. In SSMS, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, navigate to **D:\Demofiles\Mod10\Demo**, click **Demo.ssmssln**, and then click **Open**.
3. In Solution Explorer, expand **Queries**, double-click **Demo 2 - database mail.sql**, review the code, and then click **Execute**.
4. View the results. The first result set shows system events for Database Mail, and the second result set shows records of email messages that have been sent.
5. Keep the solution and SQL Server Management Studio open for the next demonstration.

Question: You are troubleshooting Database Mail. You want to see a list of the email messages that have been successfully sent and a list of email messages that could not be sent. Where can you find this information?

Lesson 3
Operators, Alerts, and Notifications

Many SQL Server systems have multiple administrators. You can use SQL Server Agent to configure operators that are associated with one or more administrators and to determine when to contact each of the operators—along with the method to use for that contact.

SQL Server can also detect many situations that might be of interest to administrators. You can configure alerts that are based on SQL Server errors or on system events—such as low disk space—and then configure SQL Server to notify you of these situations.

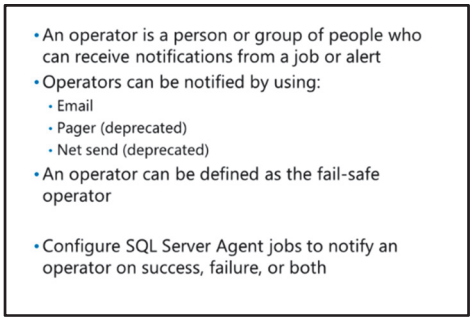
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the role of operators in SQL Server Agent.
- Describe SQL Server alerts.
- Create alerts.
- Configure alert actions.
- Troubleshoot alerts and notifications.

SQL Server Agent Operator Overview

An operator in SQL Server Agent is an alias for a person or a group of people who can receive electronic notifications when jobs complete, or when alerts are raised.



Note: Operators do not need to be Windows logins, SQL Server logins, or database users.

Configuring Operators

You can define new operators using either SSMS or the **dbo.sp_add_operator** system stored procedure. After you define an operator, you can view the definition by querying the **dbo.sysoperators** system table in the **msdb** database.

You can configure three types of contact methods for each operator:

- **Email**. An SMTP email address where notifications are sent. Where possible, you should use group email addresses rather than individual ones. You can also list multiple email addresses by separating them with a semicolon.
- **Pager email**. An SMTP email address where a message can be sent at specified times (and days) during a week.
- **Net send address**. Messenger address where a network message is sent.

 **Note:** Pager and Net send notifications are marked for deprecation, and should not be used for new development because they will be removed in a future version of SQL Server.

Fail-Safe Operator

You can also define a fail-safe operator that is notified in the following circumstances:

- The SQL Server Agent cannot access the tables that contain settings for operators and notifications in the **msdb** database.
- A pager notification must be sent at a time when no operators configured to receive pager alerts are on duty.

Job Notifications

You can configure SQL Server Agent jobs to send messages to an operator on completion, failure, or success of a job. Configuring jobs to send notifications on completing or success might lead to a large volume of email notifications, so DBAs might prefer to be advised only if a job fails. However, for business-critical jobs, you might want to be notified, regardless of the outcome, to remove any doubt over the notification system itself.

For more information about creating operators, see the topic *Create an Operator* in the SQL Server Technical Documentation:

 **Create an Operator**

https://aka.ms/A7tfri

For more information about configuring notifications for the status of SQL Server Agent jobs, see the topic *Notify an Operator of Job Status* in the SQL Server Technical Documentation:

 **Notify an Operator of Job Status**

https://aka.ms/Oeg6lt

Demonstration: Configuring SQL Server Agent Operators

In this demonstration, you will see how to:

- Enable a SQL Server Agent mail profile.
- Create an operator.
- Configure a job to notify an operator.

Demonstration Steps

Enable a Mail Profile for SQL Server Agent

1. In Object Explorer, under **MIA-SQL**, right-click **SQL Server Agent**, and then click **Properties**.
2. In the **SQL Server Agent Properties** dialog box, on the **Alert System** page, select **Enable mail profile**.
3. In the **Mail profile** drop-down list, select **SQL Server Agent Profile**, and then click **OK**.
4. In Object Explorer, right-click **SQL Server Agent**, and then click **Restart**.
5. In the **User Account Control** dialog box, click **Yes**.

6. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.

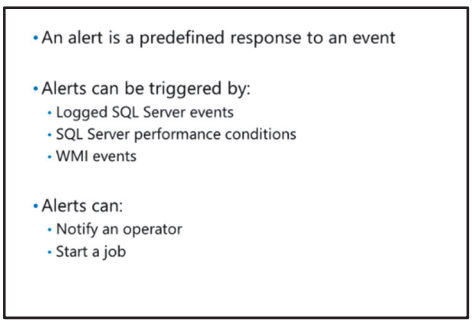
Create an Operator

1. In Solution Explorer, double-click **Demo 3 - operators.sql**.
2. Select the code under the comment that begins **Task 2**, and then click **Execute** to create a new operator called **Student**.

Configure a Job to Notify an Operator

1. In Object Explorer, expand **SQL Server Agent**, expand **Jobs** and view the existing jobs.
2. Right-click **Back Up Database - AdventureWorks**, and click **Properties**.
3. In the **Job Properties - Back Up Database - AdventureWorks** dialog box, on the **Notifications** page, select **E-mail**.
4. In the first drop-down list, click **Student**.
5. In the second drop-down list, click **When the job completes**, and then click **OK**.
6. In Object Explorer, expand **Operators**, right-click **Student**, and then click **Properties**.
7. In the **Student Properties** dialog box, on the **Notifications** page, click **Jobs**, note the job notifications that have been defined for this operator, and then click **Cancel**.
8. Under **Jobs**, right-click **Back Up Database - AdventureWorks**, and click **Start Job at Step**.
9. When the job has completed, click **Close**.
10. Under **Operators**, right-click **Student**, and then click **Properties**.
11. In the **Student Properties** dialog box, on the **History** page, note the most recent notification by email attempt, and then click **Cancel**.
12. In File Explorer, in the **C:\inetpub\mailroot\Drop** folder, verify that a new email message has been created.
13. Double-click the most recent file to view it in Outlook. Then, when you have read the message, close it, and minimize the **Drop** window.
14. Keep the solution and SQL Server Management Studio open for the next demonstration.

Overview of SQL Server Alerts



There are many events, other than scheduled jobs occurring in a SQL Server system, that are of interest to administrators. An alert is a SQL Server object defining a condition that requires attention, and a response that should be taken when the event occurs. You can define alerts to execute a job or to notify an operator when a particular event occurs, or even when a performance threshold is exceeded.

SQL Server generates events and enters them into the Windows Application Event Log. On startup, SQL Server Agent registers itself as a callback service with the Application Log. This means that the Application Log will automatically notify SQL Server

Agent when events of interest occur. This callback mechanism operates efficiently because SQL Server Agent does not need to continuously read (or poll) the Application Log to find events of interest.

When the Application Log notifies SQL Server Agent of a logged event, SQL Server Agent compares the event to the alerts that you have defined. When SQL Server Agent finds a match, it fires the alert, which is an automated response to an event.

Note: You must configure SQL Server Agent to write messages to the Windows Application Event Log if they are to be used for SQL Server Agent alerts.

Alerts Actions

You can create alerts to respond to individual error numbers or to all errors of a specific severity level. You can define the alert for all databases or for a specific database. You can also define the time delay between responses.

Best Practice: It is considered good practice to configure notifications for all error messages with severity level 19 and above.

System Events

In addition to monitoring SQL Server events, SQL Server Agent can also check conditions that are detected by Windows Management Instrumentation (WMI) events. The WMI Query Language (WQL) queries that retrieve the performance data execute several times each minute, so it can take a few seconds for these alerts to fire. You can also configure performance condition alerts on any of the performance counters that SQL Server exposes.

For more information about configuring alerts, see the topic *Alerts* in the SQL Server Technical Documentation:

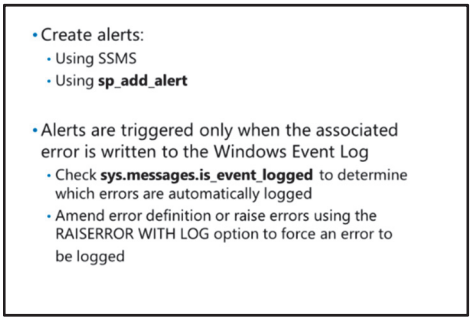
Alerts

https://aka.ms/Wixjse

Creating Alerts

You can create alerts by using SSMS or by calling the **dbo.sp_add_alert** system stored procedure. When defining an alert, you can also specify a SQL Server Agent job to start when the alert occurs.

The following example shows how to create an alert that will respond to an error message with an id of 9002 (transaction log full):



Using sp_add_alert

```
EXEC msdb.dbo.sp_add_alert
 @name=N'AdventureWorks Transaction Log
Full',
 @message_id=9002, @delay_between_responses=0,
 @database_name=N'AdventureWorks';
GO
```

Logged Events

You have seen that alerts will only fire for SQL Server errors if the error messages are written to the Windows Application Event Log. In general, error severity levels from 19 to 25 are automatically written to the Application Log but this is not always the case. To check which messages are automatically written to the log, you can query the **is_event_logged** column in the **sys.messages** table.

Most events with severity levels less than 19 will only trigger alerts if you perform one of the following steps:

- Modify the error message by using the **dbo.sp_altermessage** system stored procedure to make it a logged message.
- Raise the error in code by using the RAISERROR WITH LOG option.
- Use the **xp_logevent** system extended stored procedure to force entries to be written to the log.

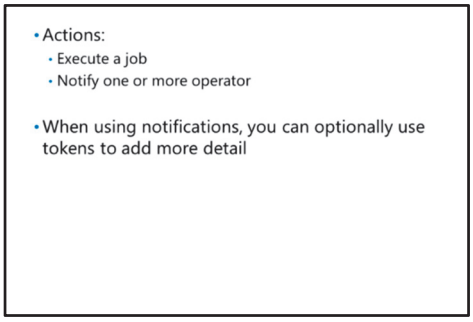
For more information about creating alerts with **sp_add_alert**, see the topic *sp_add_alert (Transact-SQL)* in the SQL Server Technical Documentation:

sp_add_alert (Transact-SQL)

https://aka.ms/Rb92a9

Configuring Alert Actions

You can configure two different actions to take in response to an alert; execute a job, and notify operators. Alert actions are optional; you can specify either, neither, or both actions for an alert.



Execute a Job

You can configure a SQL Server Agent job to execute in response to an alert. If you need to start multiple jobs, you must create a new one that starts each of your multiple jobs in turn, and then configure an alert response to run the new job.

Notify Operators

You can define a list of operators to notify in response to an alert by running the **dbo.sp_add_notification** system stored procedure. When sending messages to operators about alerts, it is important to provide the operator with sufficient context so that they can determine the appropriate action to take.

You can include tokens in the message to add detail. There are special tokens available for working with alerts, including:

- **A-DBN**. Database name.
- **A-SVR**. Server name.
- **A-ERR**. Error number.
- **A-SEV**. Error severity.
- **A-MSG**. Error message.

By default, the inclusion of tokens is disabled for security reasons, but you can enable it in the properties of SQL Server Agent.

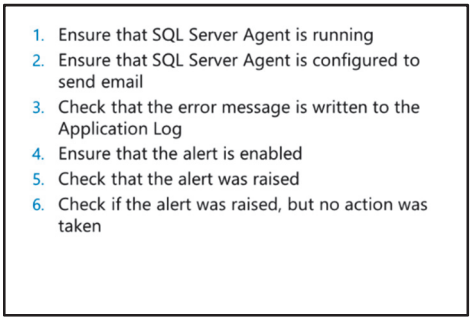
For more information about using tokens in SQL Server Agent notifications, see the topic *Use Tokens in Job Steps* in the SQL Server Technical Documentation:

Use Tokens in Job Steps

https://aka.ms/B8e4hn

Troubleshooting Alerts and Notifications

When troubleshooting alerts and notifications, use the following process to identify the issues:



1. **Ensure that SQL Server Agent is running**. The Application Log will only send messages to SQL Server Agent when the agent is running. The Application Log does not hold a queue of notifications to make at a later date.
2. **Ensure that SQL Server Agent is configured to send email**. If SQL Server Agent is not configured with a Database Mail profile, no notifications will be sent, even if all other aspects of the notification system are configured.
3. **Check that the error message is written to the Application Log**. For SQL Server event alerts, check that the error message is written to the Application Log. You should also make sure that the Application Log is configured with sufficient size to hold all the event log details.
4. **Ensure that the alert is enabled**. If the alert is disabled, it will not fire.
5. **Check that the alert was raised**. If the alert does not appear to be raised, make sure that the setting for the delay between responses is not set to too high a value.
6. **Check if the alert was raised, but no action was taken**. Check that the job is configured to respond to the alert functions as expected. For operator notifications, check that Database Mail is working and that the SMTP server configuration is correct. Test the Database Mail profile that is sending the notifications by manually sending mail from the profile used by SQL Server Agent.

Demonstration: Configuring SQL Server Agent Alerts

In this demonstration, you will see how to:

- Create an alert.
- Test an alert.

Demonstration Steps

Create an Alert

1. In Object Explorer, under **SQL Server Agent**, right-click **Alerts**, and then click **New Alert**.
2. In the **New Alert** dialog box, on the **General** page, in the **Name** box, type **Log Full Alert**.

3. In the **Type** drop-down list, note that you can configure alerts on WMI events, performance monitor conditions, and SQL Server events, and then click **SQL Server event alert**.
4. Click **Error number**, and in the box, type **9002** (which is the error number raised by SQL Server when a database transaction log is full).
5. On the **Response** page, select **Notify operators**, and then select the **E-mail** check box for the **Student** operator.
6. On the **Options** page, under **Include alert error text in**, select **E-mail**, and then click **OK**.

Test an Alert

1. In Solution Explorer, double-click **Demo 4 - alerts.sql**, and then click **Execute**. Wait while the script fills a table in the **TestAlertDB** database. When the log file for that database is full, error 9002 occurs.
2. In Object Explorer, expand **Alerts**, right-click **Log Full Alert**, and then click **Properties**.
3. In the **'Log Full Alert' alert properties** dialog box, on the **History** page, note the **Date of last alert** and **Date of last response** values, and then click **Cancel**.
4. In File Explorer, in the **C:\inetpub\mailroot\Drop** folder, verify that a new email message has been created.
5. Double-click the most recent message to view it in Outlook. Then, when you have read the message, close it, and close the **Drop** window.
6. Leave the solution and SSMS open for the next demonstration.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? SQL Server Database Mail can only be used for sending alerts and notifications.	

Lesson 4
Alerts in Azure SQL Database

The alerting system used in Azure SQL Database is different from the operators, alerts, and notifications system used by on-premises SQL Server instances. This lesson covers how to configure and work with alerts in Azure SQL Database.

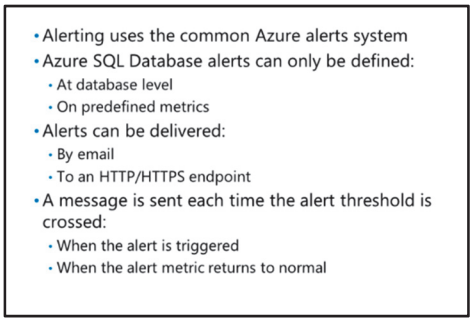
Lesson Objectives

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

- Describe the alerting system in Azure SQL Database, and how it differs from a SQL Server instance.
- Configure alerts in Azure SQL Database.

Alerts in Azure SQL Database Overview

When you configure notifications and alerts in Azure SQL Database, you do not configure operators and alerts as you do for a SQL Server instance. Instead, you use the Azure notification system, which is common across all Azure services.



As you saw in the previous lesson, when you configure an alert on a SQL Server instance, you configure an alert at server level or database level—and you can choose to trigger the alert, based on a specific error number or a specific error severity. Alerts in Azure SQL Database are created at database level, and can be configured on a list of predefined metrics. This list includes:

- Blocked by firewall
- Failed connections
- Successful connections
- CPU percentage
- Deadlocks
- DTU percentage
- DTU limit
- DTU used
- Log IO percentage
- Data IO percentage
- Sessions percentage
- Total database size
- Database size percentage
- Workers percentage

Note: Metrics can be added to or removed from this list when Azure SQL Database features are added, removed, or changed. There is no facility to use Database Engine error messages as the basis for an alert outside the list of predefined metrics.

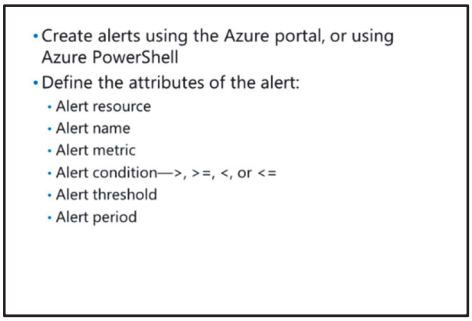
After a metric is selected, you configure an alert threshold and a reporting interval, and a list of email addresses to notify. You can also deliver the alert to an HTTP or HTTPS endpoint using Azure webhooks.

Note: Azure alerts are sent from within the Azure cloud. If network connectivity is not available between your location and the Azure cloud—either because of a network outage or because of missing firewall rules—you will not receive alerts from the Azure infrastructure.

Each alert is made up of two messages: one message that indicates when the alert threshold has been crossed, and a second message sent when the alert has resolved, and the metric has returned to a nonalert state.

Configuring Alerts in Azure SQL Database

You define Azure SQL Database alerts using the Azure portal, or using Azure PowerShell™ commands.



To configure an alert, you define the following attributes:

- The database to which the alert applies—referenced as the *resource*.
- An alert name.
- An alert description (optional).
- The metric on which the alert is based. Note that the aggregation used in the alert threshold is determined by the metric—either a count, a byte count, or a percentage.
- The condition that triggers the alert—one of greater than, greater than or equal to, less than, or less than or equal to.
- The threshold value which triggers the alert. This is used in combination with the condition to determine when the alert is triggered.
- The alert period. The frequency at which the Azure alert system polls the metric value—preconfigured intervals between five minutes and four hours. This setting determines the greatest frequency with which you will receive the alert.
- Whether to alert owners, contributors to, and readers of the database. By default, this will notify the administrator of the Azure subscription under which the Azure SQL Database is running.
- Additional email addresses to notify (optional)—in a semicolon-delimited list.
- The webhook endpoint to notify (optional).

For more information about configuring Azure alerts through the Azure portal, see the topic *Create alerts* in the Azure documentation:

Create alerts

https://aka.ms/Ixkogu

For more information about adding Azure alerts using Azure PowerShell, see the command *Add-AzureRmMetricAlertRule* in the Azure documentation (these alerts are not database specific):

Add-AzureRmMetricAlertRule

https://aka.ms/Mvgiio

For more information about using webhooks to deliver Azure alerts, see the topic *Configure a webhook on an Azure metric alert* in the Azure documentation:

Configure a webhook on an Azure metric alert

https://aka.ms/je84eu

Demonstration: Configuring Alerts in Azure SQL Database

In this demonstration, you will see how to configure an Azure SQL Database alert.

Demonstration Steps

1. Open Internet Explorer, and browse to **https://portal.azure.com/**.
2. Sign in to the Azure portal with your Azure pass or Microsoft account credentials.
3. In the menu, click **SQL databases**, and then on the **SQL databases** blade, click **AdventureWorksLT**.
4. On the **Settings** blade, under **Monitoring**, click **Alert rules**.
5. On the **Alert rules** blade, click **Add alert**.
6. On the **Add an alert rule** blade, notice that the **Resource** box is automatically populated with the database name.
7. Configure the alert using the following values, and then click **OK**:
   - **Name**: AdventureWorksLT DTU usage alert.
   - **Metric**: DTU percentage.
   - **Condition**: greater than.
   - **Threshold**: 1.
   - **Period**: Over the last 5 minutes.
   - **Additional administrator email(s):** Add your email address to receive the alert.
8. In SSMS, in Solution Explorer, double-click **Demo 5 - azure.sql**.
9. On the **Query** menu, point to **Connection**, and then click **Change Connection**.

10. In the **Connect to Database Engine** dialog box, connect to the Azure SQL Database server hosting your copy of the **AdventureWorksLT** database. (You must use the credentials you configured when you configured the Azure SQL Database server for this connection. If you are unsure of the server name, it is shown on the **AdventureWorksLT** blade of the Azure portal.)
11. On the toolbar, in the **Available Databases** list, click **AdventureWorksLT**.
12. On the toolbar, click **Execute**. The query executes a simple SELECT statement 200 times, which should raise DTU consumption on the database above 1 percent.
13. In the Azure portal, wait for the **Alert rules** blade to update so that the rule shows a **LAST ACTIVE** value other than **Never**, indicating that the alert was triggered. This could take several minutes and you might need to refresh the page.
14. On the **Alert rules** blade, click **AdventureWorksLT DTU usage alert**. Notice that a line chart shows the alert metric over time, with the threshold marked as a dotted line.
15. Click **Delete**, then click **Yes** to remove the rule.
16. If you are planning to show the content of an alert email message, log into your mailbox and examine mail delivered from **Microsoft Azure Alerts**.
17. Close Internet Explorer, and then close SSMS without saving any changes.

Check Your Knowledge

Question	
Which of the following metrics cannot be used as the basis for an Azure SQL Database alert?	
Select the correct answer.	
	DTU percentage
	SQL Server error number 9002
	Total database size
	CPU percentage
	Blocked by Firewall

Lab: Monitoring SQL Server with Alerts and Notifications

Scenario

You are a database administrator (DBA) at Adventure Works Cycles with responsibility for the **AWDataWarehouse**, **HumanResources**, and **InternetSales** databases. You have configured jobs to automate backups of these databases, and now you want to implement notifications and alerts to help you proactively manage the database.

Objectives

After completing this lab, you will be able to:

- Configure Database Mail.
- Configure operators.
- Configure alerts and notifications.
- Test alerts and notifications.

Estimated Time: 60 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Configuring Database Mail

Scenario

All database administrators at Adventure Works use email messages as a primary means of communication. You therefore plan to use Database Mail to enable email notifications from SQL Server.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Configure Database Mail

3. Test Database Mail

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab10\Starter** folder as Administrator.

▶ Task 2: Configure Database Mail

- Configure Database Mail in the **MIA-SQL** instance of SQL Server to add a new profile named **SQL Server Agent Profile** with the following account:
    - **Account name**: AdventureWorks Administrator
    - **Email address**: administrator@adventureworks.msft
    - **Display name**: Administrator (AdventureWorks)
    - **Reply email**: administrator@adventureworks.msft
    - **Server name**: mia-sql.adventureworks.msft

The new profile should be public, and it should be the default Database Mail profile.

▶ Task 3: Test Database Mail

1. Send a test email message from the Database Mail service to **student@adventureworks.msft**.
2. Verify that the test email message is successfully delivered to the **C:\inetpub\mailroot\Drop** folder.
3. Query the **dbo.sysmail_event_log** and **dbo.sysmail_mailitems** tables in the **msdb** database to view Database Mail events and email history.

Results: After this exercise, you should have configured Database Mail with a new profile named **SQL Server Agent Profile**.

Exercise 2: Configuring Operators

Scenario

Now that you have configured Database Mail, you must create operators to receive notifications. You want to receive all notifications that concern the databases for which you are responsible at your student@adventureworks.msft email address. However, you also need to configure a fail-safe operator so that notifications are sent to the DBA team alias (dba@adventureworks.msft) if there is a problem with the notification system. You also need to configure SQL Server Agent to use a Database Mail profile.

The main tasks for this exercise are as follows:

1. Create the Student Operator

2. Create the DBA Team Operator

3. Configure the SQL Server Agent Mail Profile

▶ Task 1: Create the Student Operator

- Create a new operator named **Student** with the email address **student@adventureworks.msft**.

▶ Task 2: Create the DBA Team Operator

- Create a second operator named **DBA Team** with the email address **dba@adventureworks.msft**.

▶ Task 3: Configure the SQL Server Agent Mail Profile

1. Configure the SQL Server Agent on **MIA-SQL** to use the **SQL Server Agent Profile** Database Mail profile.
2. Set the fail-safe operator to the **DBA Team** operator.
3. Restart the SQL Server Agent service.

Results: After this exercise, you should have created operators named **Student** and **DBA Team**, and configured the SQL Server Agent service to use the **SQL Server Agent Profile** Database Mail profile.

Exercise 3: Configuring Alerts and Notifications

Scenario

The **InternetSales** database is business critical, and you want to ensure that it remains operational should its transaction log become full. You therefore want to configure an alert that will notify you if the log becomes full, and will automatically run a job to back up and truncate the transaction log, keeping the database online.

You need to be notified if the jobs that back up the **AWDataWarehouse** and **HumanResources** database fail. The **InternetSales** database is critical to the business, so for peace of mind you want to be notified when the jobs to back up this database and its transaction log complete—regardless of the outcome.

The main tasks for this exercise are as follows:

1. Create an Alert

2. Configure Job Notifications

▶ Task 1: Create an Alert

1. Create an alert named **InternetSales Log Full Alert**.

2. Configure the alert to run the **Backup Log - InternetSales** job and send an email message that includes the error message to the **Student** operator if error number **9002** occurs in the **InternetSales** database.

▶ Task 2: Configure Job Notifications

1. Configure the **Back Up Database -AWDataWarehouse** and **Back Up Database - HumanResources** jobs to notify the **Student** operator on failure.

2. Configure the **Back Up Database -InternetSales** and **Back Up Log - InternetSales** jobs to notify the **Student** operator on completion.

3. Verify the job notifications assigned to the **Student** operator by viewing its notification properties.

Results: After this exercise, you should have created an alert named **InternetSales Log Full Alert**.

Also, you should have configured the **Back Up Database - AWDataWarehouse**, **Back Up Database - HumanResources**, **Back Up Database - InternetSales**, and **Back Up Log - InternetSales** jobs to send notifications.

Exercise 4: Testing Alerts and Notifications

Scenario

Now that you have configured alerts and notifications, you will test them by manually triggering jobs, and then filling the transaction log of the **Internet Sales** database.

The main tasks for this exercise are as follows:

1. Test the Alert

2. Test Job Notifications

▶ Task 1: Test the Alert

1. Use the **TestAlert.sql** script in the **D:\Labfiles\Lab10\Starter** folder to fill the log in the **InternetSales** database.
2. View the history properties of the **InternetSales Log Full Alert** alert to verify the most recent alert and response.
3. Verify that notification email messages for the full transaction log error and the completion of the **Back Up Log - InternetSales** job were successfully delivered to the **C:\inetpub\mailroot\Drop** folder.

▶ Task 2: Test Job Notifications

1. Run the **Back Up Database - AWDataWarehouse** job (which should fail), and the **Back Up Database - HumanResources** and **Back Up Database - InternetSales** jobs (which should succeed).
2. View the history properties of the **Student** operator to verify the most recent notification that was sent.
3. Verify that notification email messages for the failure of the **Backup Database - AWDataWarehouse** job and the completion of the **Back Up Database - InternetSales** job were successfully delivered to the **C:\inetpub\mailroot\Drop** folder.

Results: After this exercise, you will have verified that the notifications you configured for backups of the **AWDataWarehouse**, **HumanResources**, and **InternetSales** databases work as expected.

You will also verify that an alert is triggered when the transaction log of the **Internet Sales** database is full.

Question: Under what circumstances would email notifications be sent to the DBA Team operator you created?

Module Review and Takeaways

In this module, you learned how SQL Server logs errors and how you can use Database Mail and the notification system supported by the SQL Server Agent to manage database servers and databases proactively.

 **Best Practice:** When using Database Mail, and planning notifications and alerts in SQL Server, consider the following best practices:

- Configure different Database Mail profiles for different usage scenarios.
- Provide limited access to the ability to send email messages from the Database Engine.
- Implement a retention policy for Database Mail log and mail auditing.
- Create a fail-safe operator.

Define alerts for severe error messages.

Review Question(s)

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? You want to designate a colleague in the IT team as an operator, but this colleague does not have a login in the SQL Server instance. You must define a login for your colleague before they can be configured as an operator.	

Question: You are planning to send notifications from SQL Server, and think it might be easier to use NET SEND notifications instead of email. Why should you not do this?

Module 11

Introduction to Managing SQL Server Using PowerShell

Contents:

Module Overview

This module looks at how to use Windows PowerShell® with Microsoft SQL Server®. Businesses are constantly having to increase the efficiency and reliability of maintaining their IT infrastructure; with PowerShell, you can improve this efficiency and reliability by creating scripts to carry out tasks. PowerShell scripts can be tested and applied multiple times to multiple servers, saving your organization both time and money.

Objectives

After completing this module, you will be able to:

- Describe the benefits of PowerShell and its fundamental concepts.
- Configure SQL Server by using PowerShell.
- Administer and maintain SQL Server by using PowerShell.
- Manage an Azure SQL Database by using PowerShell.

Lesson 1
Getting Started with Windows PowerShell

Windows PowerShell includes a shell—or console—and a scripting language. You can use it with SQL Server to automate common SQL Server tasks. In this lesson, you will learn about how PowerShell works and how it supports working with SQL Server.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain what PowerShell is, and when you would use it.
- Use PowerShell help.
- Access the PowerShell console and Integrated Scripting Environment (ISE).
- Explain what SQL Server Management Objects are.
- Describe PowerShell providers.

What Is Windows PowerShell?

Windows PowerShell is an object-oriented scripting language that includes an extensive library of common commands, known as cmdlets. You can also write your own cmdlets.

- Used to automate Windows tasks and configuration settings, including SQL Server
- Uses commands, or cmdlets, in the format verb+prefixnoun (noun is always singular)
- Is case insensitive
- Includes aliases, or shortcuts, such as **cd**, **cls**, and **dir**
- Accessed either from the taskbar or Object Explorer

Why PowerShell Was Developed

Why use PowerShell? Windows is graphical, friendly, and colorful—who would want to go back to a text-based, scripting language? The answer is simple: PowerShell scripts are scalable, repeatable, and work across many Windows products, including operating systems. This is a very powerful concept because it means you can automate tasks that include the operating system, in addition to SQL Server.

With PowerShell, you can create scripts that work on one machine, 10 machines, or all the machines in your data center. You can then pass the scripts to someone else to run, with the confidence that they will obtain the same results. You cannot do that with a graphical user interface (GUI)—each time you perform a task using the GUI, it takes about the same time; if you ask someone else to do the job, they may or may not do it the same way. The GUI may be friendly, but it can be time consuming for frequently-run tasks. With PowerShell, you can automate tasks using scripts, saving time and improving quality.

PowerShell cmdlets

cmdlets take the form of a verb plus a prefix noun; for example, Get-Help, or Import-Module, or Get-Verb. Although PowerShell is case-insensitive, the code is more readable when it's capitalized appropriately. The prefix denotes the PowerShell module; for example, SQL Server PowerShell has a prefix of sqlps. All the SQL Server PowerShell cmdlets can be displayed by typing **Get-Command -Module sqlps**. If you want to display all the cmdlets, you type **Get-Command**.

PowerShell verbs are standardized to help you become familiar enough to guess commands for a given situation.

Note: The noun is always singular in the verb+prefixnoun syntax. This makes commands easier to remember.

PowerShell Aliases

If the cmdlet format seems a little unfamiliar, there are some commands that anyone with command-line or Unix experience will recognize. You can, for example, use **cd** to change directories, **cls** to clear the screen, or **dir** to display a list of directories. These are not actually PowerShell commands, but aliases. You can also think of them as shortcuts to some common commands. You can display a list of aliases by typing **Get-Alias**.

PowerShell and SQL Server

You can use PowerShell to automate common SQL Server tasks and configuration settings. Running within a command-line shell, this makes it possible for you to carry out tasks on both local machines and remote servers. A GUI may be friendly if you only have to carry out a task on one machine, but it is not efficient if that same task has to be carried out on several machines. PowerShell scripts can be developed, tested, and then run multiple times with predictable results.

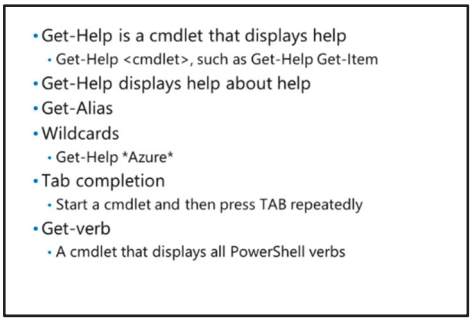
PowerShell Console

You can start PowerShell either from the taskbar or directly from Object Explorer in SQL Server Management Studio. Whichever method you use, the PowerShell console opens.

Note: Many cmdlets will not run unless you run PowerShell as Administrator. When you start the PowerShell console as Administrator, the top title bar reads: *Administrator: Windows PowerShell*. If you see a different title, you know you have forgotten to run in Administrator mode.

PowerShell Help

A good place to start when learning PowerShell is the Get-Help cmdlet. This cmdlet displays a page that explains how to get more information, including downloading or updating help files.



Download Help for First Time Use

Before using PowerShell help, you must specifically download it. When you access help, you might be prompted to download updated help for the first time or to update help files from the Internet when updates are available.

Note: you must be running the PowerShell console as Administrator to download the files.

Get-Help Cmdlet

When you understand how to use PowerShell help, learning PowerShell becomes a lot easier. You have already seen that the cmdlet, Get-Help, displays a summary of the help available in PowerShell. Within this summary, you will learn how to access help for specific cmdlets.

Use Help to find the syntax for the Get-Item cmdlet:

Get-Help

```
Get-Help Get-Item
```

The Get-Help cmdlet returns the name of the specified cmdlet, a synopsis of what it does, the syntax, description, related links, and remarks. Within the syntax section, there might be one or more syntax sections because a cmdlet can take different parameters. For example, **Get-Item** may have **-Path**, **-Credential**, or **-Stream** in position one.

Get Help About Get-Help

Get-Help is a cmdlet, and like all cmdlets, you can obtain help about the Get-Help cmdlet. If that sounds circuitous, it is also very helpful. If you type **Get-Help Get-Help**, you will learn that you can specify **-Show Window**, **-Parameter**, **-Online**, **-Detailed**, or **-Examples** as parameters. **-ShowWindow** opens help in a window and **-Examples** gives a list of examples for the cmdlet.

Get-Alias

The **Get-Alias** cmdlet displays a list of all aliases, including **Gal**, which is an alias for **Get-Alias**. Use **Get-Alias** to explore shortcuts to commonly used cmdlets; for example, **Get-Alias Get-Help**.

Wildcards

You can use wildcards to discover cmdlets related to a particular task. If you want to display all the cmdlets that start with the Get verb, type **Get-Help Get***. If you want to display all the cmdlets related to SQL Server, type **Get-Help *SQL*** or **Get-Help *SQLServer***. You can discover the cmdlets related to Azure by typing **Get-Help *Azure***.

Tab Completion

Using tab completion, you can type the first part of a cmdlet, and PowerShell will complete it for you. It does not just complete the cmdlet, but also capitalizes it for you. Furthermore, you can type part of a cmdlet, and then press TAB repeatedly to see all the cmdlets that begin in that way.

If you type **Get-I** and press TAB repeatedly, you will see all the cmdlets that start with Get-I, such as **Get-Item**, **Get-ItemProperty**, **Get-InitiatorId**. Tab completion saves time, saves mistakes, and makes it possible for you to see the range of cmdlets starting with a particular verb. If you mistakenly go past the cmdlet you want, use SHIFT+TAB to go back.

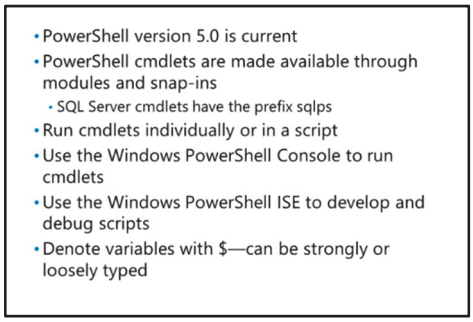
Explore Further

With help, you can work faster and do more with PowerShell. It is worth spending some time to understand the PowerShell help system, and practice using it. Then when you need to do something, you will know how to explore the cmdlets and find what you want.

 **Note:** The cmdlet **Get-Verb** lists all PowerShell verbs. Type **Get-Help Get-Verb** for more information.

Getting Started with PowerShell

PowerShell was first released in 2009, with the latest version, PowerShell 5.0, released in February 2016. This course uses PowerShell version 4.0. You can display the installed version by using the command, **$psversiontable**.



PowerShell Modules

PowerShell commands are installed in modules or snap-ins. Each cmdlet is made up of a verb-prefix-noun. The prefix denotes the module, so sqlps is the prefix for SQL Server cmdlets, and azurerm is the prefix for Azure Resource Manager cmdlets. There are also modules for Active Directory (ad), FTP (psftp), file security (acl), and many more.

PowerShell loads the modules it requires into the active PowerShell console, either when you type a cmdlet, or explicitly when you type **import-module sqlps**.

Console or ISE

You can use PowerShell in one of two ways, either through the console or the PowerShell ISE (Integrated Scripting Environment).

The console is started from the taskbar, and can be used to interact directly with the remote server by typing cmdlets and viewing the information that is returned. Use the console when you want information back from the server, and for testing scripts.

The ISE is designed to create, edit, debug, and run scripts. It can be started from the **Windows PowerShell** group on the Start menu. You can also run cmdlets in the lower pane of the ISE.

PowerShell Scripts

PowerShell scripts are saved with a .ps1 extension. PowerShell scripts must be opened explicitly: they do not respond to double-clicking.

PowerShell Variables

Variables are identified with a $ symbol, and may be loosely or strongly typed. That is to say, you may either declare a variable that will accept any type of data, such as text or numbers—or you can declare a variable that will only take one data type. The advantage of strongly typed variables—which only accept one data type—is that you can be more confident of results. Loosely typed variables may seem more flexible, but are a common source of bugs in systems, and can be difficult to spot.

You declare a variable by typing **$variablename**, where variable name is the name of your variable. You then assign a value using the equal sign. If you want to declare a variable called DatabaseName and assign it a value of AdventureWorks2016, type the following:

Declaring Loosely Typed Variables

```
$databaseName = "AdventureWorks2016"
```

If you want to declare a strongly typed variable, type the following:

Declaring Strongly Typed Variables

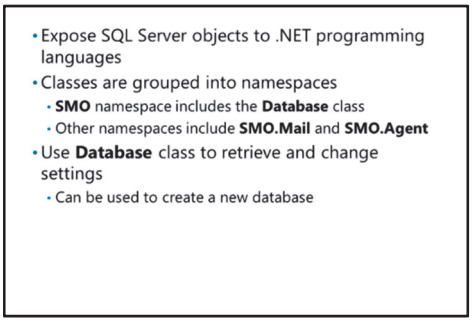
```
[int]$databaseCount = 3
```

There are different data types you can use, including the following common types:

- [int] – integers
- [string] – characters
- [char] – one character
- [xml] – an xml document

SQL Server Management Objects

SQL Server Management Objects (SMOs) were introduced with SQL Server 2005 as a way of exposing SQL Server internal objects to .NET programming languages. With PowerShell, you can use SMOs to retrieve and configure SQL Server settings, create new databases, and more. SMO utility classes are used to carry out other SQL Server tasks, such as backups and restores.



SMO Namespaces

SMO is object-oriented and so organizes its classes in namespaces. The classes used to manipulate server and database settings are in the SMO namespace. Other namespaces are **SMO.Agent**, which represents SQL Server Agents, and **SMO.Mail**, which represents Database Mail. For a list of all SMO namespaces, together with links to their classes, see the SQL Server Technical documentation:

SMO Object Model Namespaces

https://aka.ms/Fixkdk

SMO Database Class

Within the **SMO** namespace, a class called **Database** represents a SQL Server database. Using PowerShell, you can use the **Database** class to get and set database properties, or to create a new database.

For an overview of SQL Server Management Objects, see MSDN:

SQL Server Management Objects (SMO) Programming Guide

https://aka.ms/Y2hn3t

Note: The SMO object model follows a similar hierarchical structure to that shown by Object Explorer in SSMS.

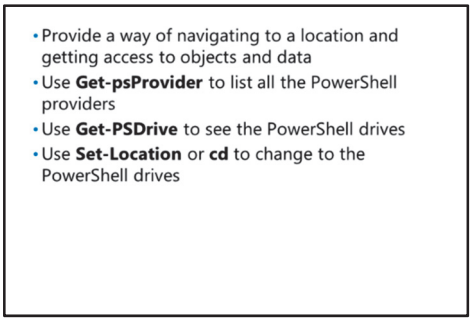
For a visual hierarchy of the SMO objects, see the MSDN:

SMO Object Model Diagram

https://aka.ms/Glt2w5

PowerShell Providers

PowerShell providers offer a method of navigating to, and accessing objects or data at, a certain location. Consider File Explorer, a familiar utility that provides a way of accessing data at certain locations on a disk. PowerShell providers are more flexible—you can use them to access not only locations such as the file system, but also other locations, such as the Windows registry. You can use providers in PowerShell to access objects or data.



Get-PSProvider

The **Get-PSProvider** cmdlet displays a list of PowerShell providers and their associated drives.

To change to a location, use the **Set-Location** cmdlet (or its alias **cd**) and type the **PSDrive** name followed by a colon:

Using Set-Location

```
Set-Location Env:
```

When you import a module, such as SQLPS, the SQL provider is also loaded and available.

Provider cmdlets

Provider cmdlets are used to access the data or objects. They include cmdlets such as **Get-PSProvider**, **Get-PSDrive** to return the current drive, or **Get-Item**, which retrieves data or objects from a location.

For a full list of PowerShell provider cmdlets, see the MSDN:

Provider Cmdlets

http://aka.ms/Od0vzt

Additional Reading: To learn more about Windows PowerShell, see *Learn Windows PowerShell in a Month of Lunches* (Don Jones and Jeffery D. Hicks). It is published by Manning and is available as an e-book.

Demonstration: Exploring SQL Server Management Objects

In this demonstration, you will see how to:

- Use PowerShell to explore the help system.
- Use a PowerShell provider to explore SMOs.

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\Mod11** folder as Administrator.
3. In the **User Account Control** dialog box, click **Yes**.

4. On the taskbar, right-click **Windows PowerShell**, and then click **Run as Administrator**.
5. In the **User Account Control** dialog box, click **Yes**.
6. In the PowerShell console, verify that the title bar reads **Administrator: Windows PowerShell**.
7. At the command prompt, type **cd **, and then press Enter
8. To understand why commands prompts appear to be working in PowerShell, at the command prompt, type **Get-Alias**, and then press Enter.
9. At the command prompt, type **Get-Alias c***, and then press Enter.
10. At the command prompt, type **Update-Help -Force -ErrorAction SilentlyContinue**, and then press Enter. Wait for the help files to install.
11. At the command prompt, type **Get-Help**, and then press Enter.
12. At the command prompt, type **Get-Help Get-Help**, and then press Enter.
13. At the command prompt, type **Get-Help Get-Item**, and then press Enter.
14. For each of the following cmdlets, at the command prompt, type the cmdlets, and then press Enter to show the different sets of parameters:

    Get-Help Get-Item -Examples
    Get-Help Get-Item -Detailed
    Get-Help Get-Item -ShowWindow
15. Close the Get-Item Help window.
16. At the command prompt, type **Get-I**, and then press TAB.
17. Press TAB repeatedly to show all the cmdlets that start with Get-I.
18. Press SHIFT+TAB to step backwards. Note how the capitalization is automatically corrected.
19. Press ESC.
20. At the command prompt, type **Get-PSProvider**, and then press Enter.
21. At the command prompt, type **Get-PSDrive**, and then press Enter.
22. At the command prompt, type **Import-Module SQLPS**, and then press Enter.
23. Repeat steps 20 and 21 and note the additions to the list.
24. At the command prompt, type **Set-Location SQLSERVER:** or **cd SQLSERVER:**, and then press Enter.
25. At the command prompt, type **Get-ChildItem**, and then press Enter.
26. At the command prompt, type **Set-Location SQL**, and then press Enter.
27. At the command prompt, type **Get-ChildItem**, and then press Enter.
28. At the command prompt, type **Set-Location MIA-SQL**, and then press Enter.
29. At the command prompt, type **Get-ChildItem**, and then press Enter.
30. At the command prompt, type **Set-Location Default**, and then press Enter.
31. At the command prompt, type **Get-ChildItem**, and then press Enter.
32. Review the list of objects and consider how they map to objects in SQL Server.
33. At the command prompt, type **Set-Location Databases**, and then press Enter.
34. At the command prompt, type **Get-ChildItem**, and then press Enter.
35. At the command prompt, type **Exit**, and then press Enter.

Check Your Knowledge

Question	
What is a PowerShell alias?	
Select the correct answer.	
	A PowerShell variable.
	The Unix alternative to PowerShell cmdlets.
	A familiar command line shortcut for a PowerShell cmdlet.
	A way of getting more information about a cmdlet.
	The full version of a cmdlet.

Lesson 2
Configure SQL Server Using PowerShell

You can use PowerShell in conjunction with SMO objects to view and amend settings, either from the PowerShell console, or from within a PowerShell script; you can build scripts that automate tasks, and apply them on many machines—or many times.

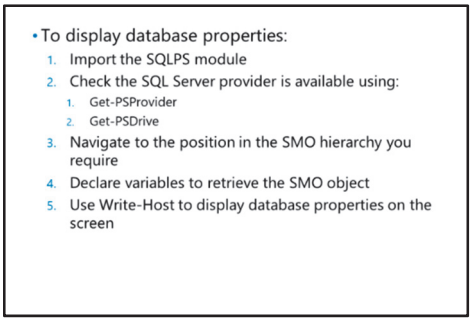
Lesson Objectives

After completing this lesson, you will be able to:

- Use PowerShell to read SQL Server database settings.
- Use PowerShell to amend SQL Server database settings.
- Use PowerShell to configure SQL Server instance settings.
- Use PowerShell ISE.

Reading Database Settings

In the previous lesson, you looked at PowerShell cmdlets, the SMO, and PowerShell providers. You can now put all these pieces together to display database properties using PowerShell.



Import SQL Module

Windows PowerShell uses the SMO so that you can display and configure database settings. To access the SMO, the SQL PowerShell module must be imported. This includes the SQL PowerShell provider.

SQL Server Provider

When the SQL PowerShell module has been loaded into the console, you can list the PowerShell providers by typing **Get-PSProvider**. Use **Get-PSDrive** to find out how to navigate to the SQL Server root. You can then navigate to the correct place in the SMO structure.

Declare Variables

The following example shows how you can use a PowerShell provider to access the SMO objects. Two variables are created: $databaseName and $database. You should assign the string value AdventureWorks2016 to $databaseName, and then use the variable to retrieve the SMO database object and assign it to the variable $database.

When you have assigned the SMO database object for AdventureWorks2016 to a variable, you can then display properties such as Name and DataSpaceUsage. These are displayed in the console using the Write-Host cmdlet.

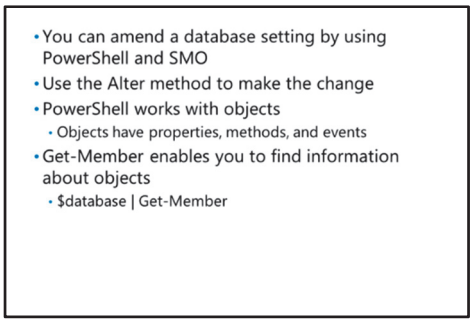
Display database properties using the SMO.Database class.

Navigating the SMO.Database Classes

```
Import-Module SQLPS;

Get-PSProvider;
Get-PSDrive;

Set-Location SqlServer:
Set-Location \SQL\localhost\DEFAULT\Databases;

$databaseName =”AdventureWorks2016”;
$database = Get-Item -Path $databaseName;
Write-host (“Database name:” ($database.Name);
Write-host(“Space used:”, ($database.DataSpaceUsage);
```

Amending SQL Server Database Settings

In this topic, you will see how to change SQL Server Database settings. You have already seen how to navigate through SMO objects using the PowerShell SQL provider, and how to assign objects to variables. You will now assign new values to the properties of objects, and then use a method to alter the setting.



Alter Method

To amend the settings, you should get the property you are interested in, and assign it a new value. To make the change permanent, use the Alter method.

This example changes the compatibility level of a database:

Calling the Alter() Method

```
# Change some of the database properties
$database.CompatibilityLevel =
[Microsoft.SqlServer.Management.Smo.CompatibilityLevel]::Version120;
$database.Alter();
Write-Host "Changed database properties";
```

Working with Objects

PowerShell is an object-oriented language that works with SQL Server objects through the SMO. When you work with SQL Server Management Studio you are not exposed to SQL Server object properties, methods, and events. They are hidden below the surface, so you can focus on managing your data.

With PowerShell, it becomes more important to understand how objects work. In the example above, you changed a property of a database object. You changed the CompatibilityLevel property, and the change was applied using the Alter method.

Get-Member

But how did you know that CompatibilityLevel was a property of the database? And how did you know that the database object had an Alter method?

Get-Member is the equivalent of Get-Help, but it applies to objects, rather than cmdlets. When you type **$database | Get-Member**, PowerShell displays a list of the object's properties, methods, and events. With Get-Member you can find out how to work with any object.

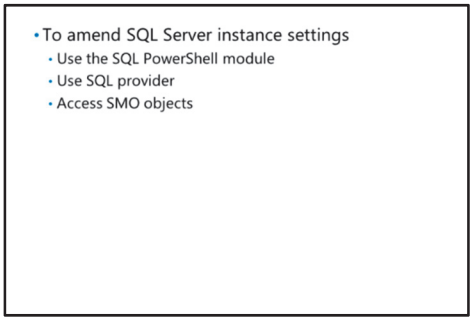
Get-Member also has an alias, **gm**, making it quick to type.

Note: Get-Member uses the pipe symbol **|** which is represented as two vertical lines on the keyboard.

Note: If you try to use Get-Member with something that is not an object, you will receive an error message.

SQL Server Settings

In addition to changing database settings, you can change SQL Server instance-level settings. In the same way as you can create a variable that represents a database object, you can create a variable to represent a server object.



SQL Server Provider

You can use the SQL PowerShell module, and the SQL provider to access SQL Server settings. With this, you can navigate to, and access, SMO objects. The server is the top level object in the SMO hierarchy; therefore, you can access the Server Settings object.

Settings Objects

In the same way as you can access the properties, methods, and events of other objects, you can use the display to amend the properties for settings. The Get-Member cmdlet displays all available properties of the settings object.

Use the Settings object to access SQL Server properties:

$server.Settings

```
$server.Settings | Get-Member -MemberType Property;
```

SQL Server Instance Settings

The SMO object settings expose some of the properties available from the **SQL Server Instance Properties** dialog box in SQL Server Management Studio (SSMS). Settings that are changed using PowerShell are reflected in the GUI of SSMS.

LoginMode

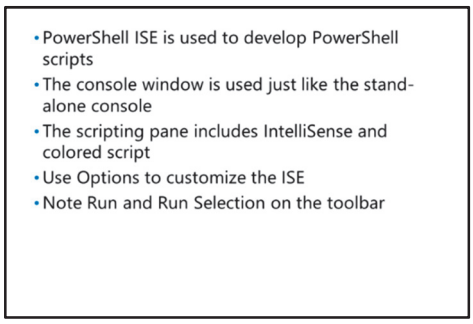
Login mode is one of the properties exposed by the settings object, and is normally set on the Security page of the **Properties** dialog box in SSMS. It can either be Windows Authentication, or SQL Server and Windows Authentication.

You can use the Settings object to amend the LoginMode property:

$server.Settings.LoginMode

```
$server.Settings.LoginMode = [Microsoft.SqlServer.Management.Smo.ServerLoginMode]::Mixed;
$server.Settings.Alter();
$server.Settings.LoginMode;
```

PowerShell ISE



The PowerShell ISE is designed to develop PowerShell scripts. It has two sections: a scripting pane where you write the script that will be saved, and the console where you can write cmdlets and display results. The console window behaves just like the stand-alone console; you can use it to test PowerShell scripts, get help about cmdlets, and make sure everything works as expected.

You should use the **View** menu to switch between displaying the scripting pane, and hiding the scripting pane.

The scripting pane includes features that make it easier to develop bug-free scripts, including IntelliSense and color coded script.

Tools, Options

You can customize ISE settings such as font size, window color, and IntelliSense. Click **Tools**, and then select **Options** to view and amend settings. It is worth getting, for example, font and font size correct for you—to make the tool easier to work with.

Toolbar

The toolbar contains many familiar icons including **new**, **open**, and **save**. It also includes two icons used to run script: the **Run** icon, which is a large solid green triangle, and the **Run Selection** icon, which is a smaller green triangle in front of a file symbol. The **Run** icon runs the entire script, and the **Run Selection** icon, runs only the code you have highlighted.

Developing Robust Scripts

Developing robust PowerShell scripts is a skill that takes time to develop. Scripts must take into account errors, objects having been deleted when you expect them to be there, and many other unexpected conditions. Script development is outside the scope of this introductory PowerShell module.

Sequencing Activity

Put the following steps in order by numbering each to indicate the correct order.

	Steps
	Import SQL PowerShell module.
	Use the SQL PowerShell provider to navigate to an SMO object.
	Assign an SMO object to a variable.
	Discover the object's properties using Get-Member.
	Amend a property.
	Apply the amendment using the Alter method.

Lesson 3
Administer and Maintain SQL Server with PowerShell

With PowerShell, you can manage a number of different Windows servers running SQL Server, using a common scripting language. This means you can automate tasks that cross operating system and SQL Server boundaries. In this lesson, you will explore a number of different tasks that introduce you to using PowerShell to administer and maintain SQL Server.

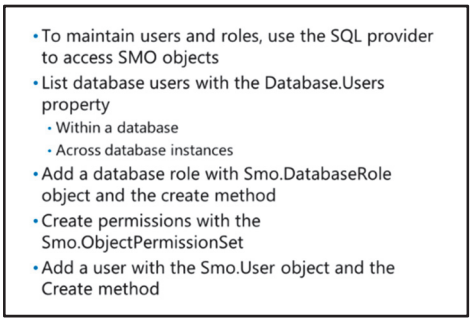
Lesson Objectives

After completing this lesson, you will be able to:

- Manage users and roles.
- Display information about SQL Server instances.
- Back up and restore databases by using PowerShell.
- Display information about Windows Updates by using PowerShell.

Managing Users and Roles

You can display and amend user settings by using PowerShell and the SMO.



Logins Property of the Server Object

To display information about users, you can use the SQL Server PowerShell module and the SQL provider to access SMO objects. Having created a variable to hold the server logins object, you can use Get-Member to find the properties, methods, and events.

Users Property of the Database Object

As before, you can create a variable to hold a database object, referring to a specific database. The database object has a property called Users, with which you can list all the database users.

Using the Users property of a database object:

Listing Specific Properties of the Users

```
$database.Users | Select-Object -Property Name, AuthenticationType, Login, LoginType,
Sid, State, DefaultSchema, CreateDate, DateLastModified, @{Name="IsMemberDbOwner";
Expression={$_.IsMember("db_owner")}} |
                Format-List;
```

Display Users from All SQL Server Instances

Accessing and amending users for a database may be useful for automating tasks using PowerShell. PowerShell can also list users from more than one SQL Server instance—something that would be difficult to do without programming.

Display users from all SQL Server instances:

Get-Children from SQL\localhost

```
Get-ChildItem | ForEach-Object { $_.Databases } |
                ForEach-Object { $_.Users } |
                Where-Object { $_.IsMember("db_owner") } |
                Select-Object -Property @{Name="ServerInstance";
Expression={$_.Parent.Parent}}, @{Name="Database"; Expression={$_.Parent.Name}}, Name,
AuthenticationType, Login, LoginType, State, DefaultSchema, @{Name="IsMemberDbOwner";
Expression={$_.IsMember("db_owner")}} |
                Format-List;
```

Add a Database Role

You can also add a database role by using the SMO object Database Role, and then calling the Create method.

Add a database role:

Smo.DatabaseRole

```
$rolename = "MyRole";

$role = New-Object Microsoft.SqlServer.Management.Smo.DatabaseRole -ArgumentList
$database, $rolename;
$role.Create();

# List the roles and make sure the new role has been created
$database.Roles;
```

Permission Sets for the Database Role

Having created the database role, you set permissions using the ObjectPermissionSet object.

Create a permission set with the Select permission:

Smo.ObjectPermissionSet

```
# The permission is passed in to the constructor of the ObjectPermissionSet object
$permset1 = New-Object
Microsoft.SqlServer.Management.Smo.ObjectPermissionSet([Microsoft.SqlServer.Management.Sm
o.ObjectPermission]::Select);

# Grant this permission set to the role you just created, so that
# it can perform select on objects in the Sales schema
$salesschema = $database.Schemas["Sales"];
$salesschema.Grant($permset1, $rolename);
```

Add a User

To add a user, you use the Smo object User, and then call the Create method.

Add a user:

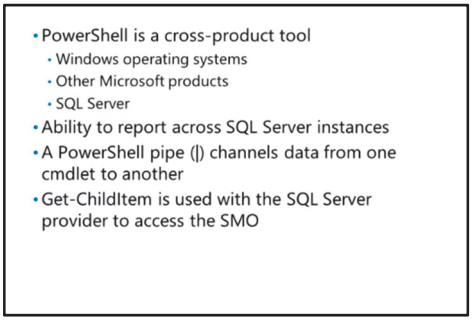
Smo.User

```
$loginname = "AW\Gragg";
$username = "Marshall";

$user = New-Object Microsoft.SqlServer.Management.Smo.User -ArgumentList $database,
$username;

$user.Login = $loginname;
$user.Create();

# List the database users and make sure the new user has been created
$database.Users;
```

This topic has introduced using the SMO hierarchy; you can access objects and properties to create and amend users, and database roles.

 **Additional Reading:** For more information about automating SQL tasks, see *SQL Server 2014 with PowerShell v5 Cookbook* (Donabel Santos, Packt Publishing).

Display Information About SQL Server Instances



A great strength of PowerShell is that it crosses boundaries. You can use PowerShell to automate tasks in Windows, other Microsoft products such as SharePoint and Exchange, and, of course, SQL Server. Using the same scripting language for everything is efficient and powerful.

Beyond Transact-SQL

You can use PowerShell in all sorts of different ways. For example, Transact-SQL works well for tasks within a database, but does not work across SQL Server instances. Yet there are times when you need an overview of SQL Server instances, perhaps to see the disk space being used, or logins that are members of the sysadmin server role.

PowerShell Pipes

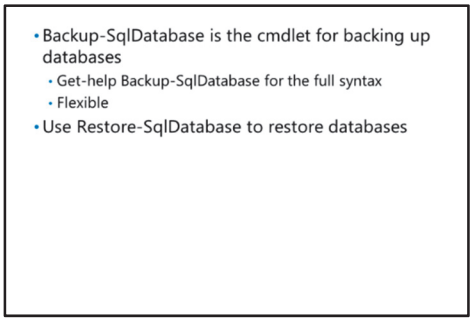
The PowerShell pipe is the | symbol. It is used to send objects from one cmdlet to another, as if they were being sent down a pipeline. To display information for several SQL Server instances, you use the **Get-ChildItem** cmdlet with pipes to display data about SQL Server instances.

Get-ChildItem

Get-ChildItem is used with the SQL Server provider to access the SMO, and retrieve information about SQL Server instances. By filtering and formatting the data, you can retrieve a useful snapshot across instances, and machines.

Managing Backups and Restores

SQL Server 2012 included updates to the sqlps PowerShell module with which you could use PowerShell to back up and restore databases.



Backup-SqlDatabase

The Backup-SqlDatabase is the cmdlet used to back up databases. It is a flexible cmdlet with seven parameter sets, meaning you can choose from a number of different ways of specifying the backup. These include the database to be backed up, the backup location, log truncation, encryption, and much more.

Type **Get-Help Backup-SqlDatabase** for the full syntax.

Use PowerShell and SMO objects to back up a database:

Backup-SqlDatabase

```
# Get the server object
# This is an SMO object and you can access its properties and methods
$server = Get-Item -Path DEFAULT;
$serverName = $server.Name;

# Get the database object
$database = $server.Databases['AdventureWorks2016']
$databaseName = $database.Name;

# Backup the database
Backup-SqlDatabase -ServerInstance $serverName -Database $DatabaseName -BackupFile
"D:\myAWBackup.bak";
```

As the database is being backed up, PowerShell shows a progress bar to indicate the backup is running.

Restore-SqlDatabase

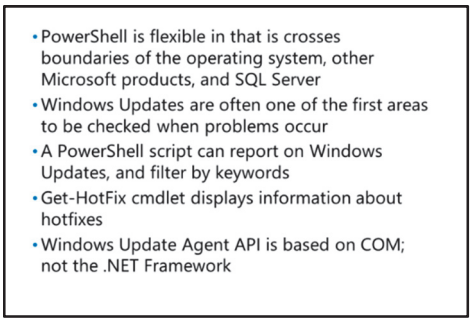
Unsurprisingly, there is an equivalent cmdlet to restore databases, namely **Restore-SQLDatabase**.

Azure Blob Storage

You can also use PowerShell to back up and restore from Azure Blob storage.

Using PowerShell to Report Windows Update

As every DBA knows, SQL Server does not operate in isolation. When problems occur, DBAs must look at what has changed within the SQL Server environment, in addition to the Windows environment. PowerShell works across Windows operating systems, in addition to SQL Server, and so is a good tool to use for troubleshooting.



Windows Updates

Windows Updates are crucial in keeping systems secure and up to date, but can also be a source of errors. Creating a PowerShell script that produces a list of Windows Updates, formatted and filtered, is a quick way of obtaining information about possible problem areas.

Windows Update PowerShell Module

The *Windows Update PowerShell* module includes cmdlets to automate and manage Windows Updates. To download the module, and for more information, see Technet:

Windows Update PowerShell Module

http://aka.ms/Dgrzrs

Get-HotFix

Get-HotFix is a PowerShell cmdlet that displays information about hotfixes that have been applied to local or remote machines. It takes parameters with which you can report on just one machine.

Hotfixes, as their name suggests, provide fixes to an urgent problem—this is a subset of Windows Updates. As such, the Get-HotFix cmdlet does not report all Windows Updates. Find out more about Get-HotFix by typing **Get-Help Get-HotFix** in the PowerShell console.

PowerShell and Windows Updates

As you have already discovered, PowerShell is a powerful, object-oriented scripting language that takes time to master. The Windows Update Agent API is based on COM, and not the .NET Framework. Because COM objects are always accessed through an interface, and not directly, you have to create a new COM object that can access the interface. In this introduction to PowerShell, you can obtain a flavor of what PowerShell is capable of—the capabilities of using PowerShell with COM are outside the scope of this module.

Demonstration: PowerShell for Troubleshooting

In this demonstration, you will see how to:

- Retrieve information about SQL Server instances.
- Output information in different formats.

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. On the taskbar, right-click **Windows PowerShell**, and then click **Run ISE as Administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. On the **File** menu, click **Open**.
5. In the **Open** dialog box, navigate to **D:\Demofiles\Mod11**, click **InspectSqlInstances.ps1**, and then click **Open**.
6. Select the code under the **#1#** comment, and then on the toolbar, click **Run Selection** to import the SQL module.
7. Select the code under the **#2#** comment, and then on the toolbar, click **Run Selection** to set the location.

8. Select the code under the **#3#** comment, and then on the toolbar, click **Run Selection** to display the instances of SQL Server.
9. Select the code under the **#4#** comment, and then on the toolbar, click **Run Selection** to display a formatted list of SQL Server instances.
10. Select the code under the **#5#** comment, and then on the toolbar, click **Run Selection** to display a list of databases in descending order of size.
11. Select the code under the **#6#** comment, and then on the toolbar, click **Run Selection** to display a tabular list of databases in descending order of size.
12. Select the code under the **#7#** comment, and then on the toolbar, click **Run Selection** to output the information to a text file.
13. Select the code under the **#8#** comment, and then on the toolbar, click **Run Selection** to output the information to an XML file.
14. Select the code under the **#9#** comment, and then on the toolbar, click **Run Selection** to output the information to an Excel file.
15. Close PowerShell ISE without saving any changes.

Lesson 4
Managing Azure SQL Databases Using PowerShell

You can use Azure PowerShell to automate a range of different tasks in Azure. Although the Azure Portal is friendly and straightforward to use, it can be time consuming if you have repetitive tasks. With Azure PowerShell, you can set up virtual machines, and automate aspects of your Azure work.

This lesson looks at how to use Azure PowerShell to connect to an existing subscription, create a new server, and then create an Azure SQL Database.

Lesson Objectives

After completing this lesson, you will be able to:

- Connect to your Azure subscription by using PowerShell.
- Create a server on Azure by using PowerShell.
- Create an Azure SQL Database by using PowerShell.
- Use Azure SQL Database cmdlets.

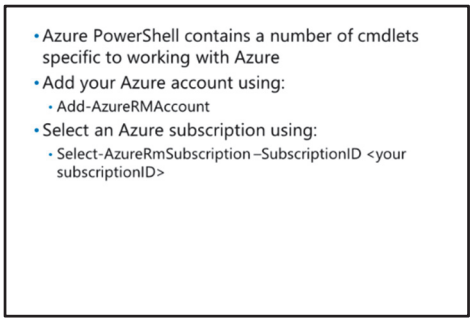
Connect to Your Subscription

There are a number of PowerShell cmdlets specific to working with Azure. These are contained in the Azure PowerShell module, which you must download and install before you can use Azure PowerShell cmdlets.

For more information, see the Microsoft Azure website:

Azure PowerShell

https://aka.ms/Kunm8d



You can get the available modules on your machine using the **Get-Module** cmdlet. You can also see available modules by using the **Get-Module -ListAvailable** cmdlet. If you have the Azure module on your machine, it will be loaded when you use one of the cmdlets.

Add Your Azure Account

To start working with Azure, you must first add your account so it can be used with Azure Resource Manager cmdlet requests. Either add your account or your login. When you successfully add your account or login, a summary of your account details is displayed in the PowerShell console.

Add your Azure account so it can be used with Azure Resource Manager.

Add Account or Login

```
Add-AzureRMAccount
Login-AzureRMAccount
```

Select Azure Subscription

As each account may have one or more subscriptions, you must specify the subscription you want to use. The subscription ID is returned when you add your Azure account or login. Alternatively, you can copy the subscription ID from the Azure Portal.

To use a specific subscription, use the **Select-AzureRmSubscription** cmdlet.

Specify a subscription:

Select-AzureRmSubscription

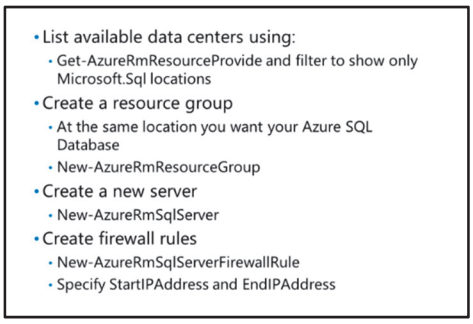
```
Select-AzureRmSubscription -SubscriptionID 927sh-ld924yh-sldhfhsk1
```

You are now ready to work with Azure PowerShell, using your Azure account, and your subscription.

Create a Server

Before you can create an Azure SQL Database, you must create a server for the database. This is done using the **New-AzureRmSqlServer** cmdlet. When you create a new server on Azure, you must specify the data center and the resource group you want to use.

The resource group is a way of keeping resources together. You must create the resource group at the same location that you want to host your Azure SQL Database.



List Azure Data Centers

When you create a server, you must provide a valid data center name. You can use the **Get-AzureRmResourceProvider** cmdlet to list the available data centers. To return the information you need, you must limit the results to only data centers that host Azure SQL Database. This is done by using a pipe to return only Microsoft.Sql locations.

List the Azure Data Centers that support Azure SQL Database.

Get-AzureRmResourceProvider

```
 (Get-AzureRmResourceProvider -ListAvailable | Where-Object {$_.ProviderNamespace -eq
'Microsoft.Sql'}).Locations
```

Create New Resource Group

You can now create a new resource group using the New-AzureRmResourceGroup cmdlet.

Create a New Resource Group.

New-AzureRmResourceGroup

```
New-AzureRmResourceGroup -Name "PowerShellTest" -Location "Japan East"
```

Create New Server

After you have created the resource group, you can create the server using the cmdlet New-AzureRmServer. The server name must be in lowercase, and must be unique throughout all the Azure data centers.

Create a new Azure SQL Database.

New-AzureRmSqlServer

```
New-AzureRmSqlServer -ResourceGroupName "PowerShellTest" -ServerName "pstest201604ce" -
Location "Japan East" -ServerVersion "12.0"
```

You will be prompted to add credentials with a user name and password.

 **Note:** Although you have to create a server for your Azure SQL Database, you do not have access to the server. You cannot log on to the server, or amend any settings. The server is required to host Azure SQL Database.

Create Firewall Rule

The last stage of creating the server is to create the firewall rule. This limits the machines that can access your Azure SQL Database to those using a range of IP addresses. Use the New-AzureRmSqlServerFirewallRule cmdlet to create a firewall rule. You can either specify a range of IP addresses, or make the StartIpAddress the same as the EndIpAddress.

Create a named firewall rule with a range of IP addresses:

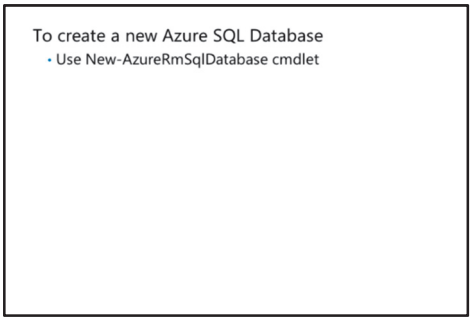
New-AzureRmSqlServerFirewallRule

```
New-AzureRmSqlServerFirewallRule -ResourceGroupName "PowerShellTest" -ServerName
"pstest201604ce" -FirewallRuleName "myFirewallRule" -StartIpAddress “nnn.nnn.nnn.nnn” -
EndIpAddress “nnn.nnn.nnn.nnn”
```

When the firewall rule has been successfully created, confirmation details are returned to the PowerShell console.

Create an Azure SQL Database

The last topic looked at how to create a server in Azure, in preparation for hosting a new Azure SQL Database. The final step is straightforward: the new database is created on the server you have already created.



The name of the database must be unique across all Azure data centers, and all letters must be in lowercase.

To create a new Azure SQL Database, you should use the New-AzureSqlDatabase cmdlet.

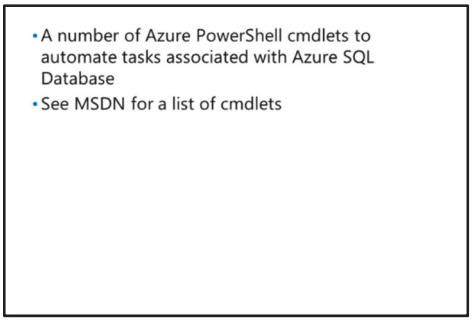
Create a New Azure SQL Database:

New-AzureRmSqlDatabase

```
New-AzureRmSqlDatabase -ResourceGroupName "PowerShell" -ServerName "pstest201604ce" -
DatabaseName "testpsdb" -Edition Standard -RequestedServiceObjectiveName "S1"
```

Azure SQL Database cmdlets

A number of cmdlets are designed to help automate tasks associated with Azure SQL Databases, some of which you have already used to create a new database.



There are cmdlets to create new items such as servers and databases. The cmdlets are divided into groups with familiar PowerShell verbs:

- Get-
- New-
- Remove-
- Set-
- Start-
- Stop-
- Suspend-
- Use-

You have already used some of these cmdlets in learning how to provision a new Azure SQL Database. For a list of Azure SQL Database cmdlets, see the MSDN:

Azure SQL Database Cmdlets

https://aka.ms/Yngoxl

Demonstration: Creating an Azure SQL Database with PowerShell

In this demonstration, you will see how to use Azure PowerShell to create an Azure SQL Database.

Demonstration Steps

Install the AzureRM PowerShell Module

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. On the taskbar, right-click **Windows PowerShell**, and then click **Run as Administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **Install-Module AzureRM**, and then press Enter.
5. At the **NuGet provider is required to continue** message, type **Y**, and then press Enter.
6. At the **Untrusted repository** message, type **A**, and then press Enter.
7. Wait until the installation completes, and then close the PowerShell window.
8. At the command prompt, type **cd**, and then press Enter.
9. At the command prompt, type **cls**, and then press Enter.

Create an Azure SQL Database

1. At the command prompt, type Link your Azure account to PowerShell, typing the following cmdlet, and then press Enter:

```
Add-AzureRmAccount
```

2. Type **Y** to enable data collection.
3. At the Azure sign-on screen, type the user name and password you use to sign in to the Azure portal, and then click **Sign in**.
4. At the command prompt, to link your Azure account to PowerShell on this VM, type the following cmdlet:

```
Login-AzureRmAccount
```

5. Wait for the Azure sign to appear on screen, type the user name and password you use to sign in to the Azure portal, and then click **Sign in**.
6. At the command prompt, type the following cmdlet, and then press Enter. Substituting the **SubscriptionID** that was returned in the previous step for <yoursubscriptionid>:

```
Select-AzureRmSubscription -SubscriptionID <yoursubscriptionid>
```

7. At the command prompt, type the following cmdlet to return a list of Azure data center locations, and then press Enter:

```
(Get-AzureRmResourceProvider -ListAvailable | Where-Object {$_.ProviderNamespace -
eq'Microsoft.Sql'}).Locations
```

8. At the command prompt, type the following cmdlet to create a resource group, and then press Enter. Substitute a location from the list returned in the previous step for <location>:

```
New-AzureRmResourceGroup -Name "20764CTest" -Location "<location>"
```

9. At the command prompt, type the following cmdlet to create a new server in the resource group you just created, and then press Enter. Substitute the location used in the previous step for <location>. Substitute a unique server name for <your server name>. This must be unique throughout the whole Azure service, so cannot be specified here. A suggested format is sql2016ps-<your initials><one or more digits>. For example, sql2016ps-js123. Letters must be lowercase.

```
New-AzureRmSqlServer -ResourceGroupName "20764CTest" -ServerName "<your server
name>" -Location "<location>" -ServerVersion "12.0"
```

10. In the **Windows PowerShell credential request** dialog box, in the **User name** box, type **psUser**, in the **Password** box, type **Pa55w.rd**, and then click **OK**. Wait for Azure to create the administrator for your server and for the console to display the information.
11. At the command prompt, type the following cmdlet to create a variable to store your external IP address and then press Enter. Substitute your own relevant information for the <your external ip> parameter:

```
$currentIP = "<your external ip>"
```

 **Note:** You can get your current external IP address from the Azure Portal (see the value returned by the "Add Client IP" button on the firewall for an existing server), or from third-party services such as www.whatismyip.com.

12. At the command prompt, type the following cmdlet to create a firewall rule that permits you to connect to the server, and then press Enter. Substitute your own relevant information for the <your server name> parameter:

```
New-AzureRmSqlServerFirewallRule -ResourceGroupName "20764CTest" -ServerName "<your
server name>" -FirewallRuleName "Firewall1" -StartIpAddress $currentIP -EndIpAddress
$currentIP
```

13. At the command prompt, type the following cmdlet to create an Azure SQL Database on the server you have just created, and then press Enter. Substitute the name of your server for <your server name>:

```
New-AzureRmSqlDatabase -ResourceGroupName "20764CTest" -ServerName "<your server
name>" -DatabaseName "testpsdb" -Edition Standard -RequestedServiceObjectiveName
"S1"
```

    This will take a few minutes to complete. Wait for the details of the new database to be returned—this indicates that the database has been created.

14. Close Windows PowerShell.

    Question: What advantages are there to using PowerShell with Microsoft Azure?

Lab: Using PowerShell to Manage SQL Server

Scenario

You are a DBA working at the quickly expanding Adventure Works Bicycle company. The IT director has just agreed to add another server and is getting concerned about how long it will take to administer all the servers. The IT director has asked you to evaluate PowerShell as a way of automating some of the tasks, and reducing the number of problems with SQL Server.

Objectives

After completing this lab, you will be able to:

- Use the PowerShell help system and providers.
- Use PowerShell to change database settings and SQL Server instance settings.

Estimated Time: 30 minutes

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

User name: **AdventureWorks\Student**

Password: **Pa55w.rd**

Exercise 1: Getting Started with PowerShell

Scenario

You have started your evaluation of PowerShell, and been advised to spend some time getting to know how the help system works. You will then investigate how to use the SQL PowerShell provider to access SMO objects.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Using PowerShell Help

3. Using PowerShell Providers

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab11\Starter** folder as Administrator.

▶ Task 2: Using PowerShell Help

1. Start Windows PowerShell as Administrator.
2. Use the Update-Help cmdlet to update the help files.
3. Display the summary help page.
4. Display online help for the **Get-ChildItem** cmdlet.
5. Get a list of all cmdlets with the **Remove-** verb.
6. Get a list of all cmdlets that include **sql** in their name.
7. Use tab-completion to view the options for the **Get-Item** cmdlet.

▶ Task 3: Using PowerShell Providers

1. List the PowerShell providers.
2. Go to the Environment location.
3. List the contents of the location.
4. List the available PSProviders.
5. List the available PSDrives.
6. Import the SQL PowerShell module.
7. List the available PSProviders again, noting the new entry in the list.
8. List the available PSDrives again, noting the new entry in the list.
9. Get help about Get-PSProvider.
10. Get help about Get-PSDrive.
11. Close the console window.

Results: After completing this exercise, you will have investigated PowerShell help and the SQL PowerShell provider.

12. Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? PowerShell providers offer an alternative to using SQL Server Management Objects.	

Exercise 2: Using PowerShell to Change SQL Server Settings

Scenario

As the Adventure Works Bicycle company grows, you want to use PowerShell to automate more SQL Server tasks. To demonstrate how PowerShell can change SQL Server settings, you develop some PowerShell scripts to show the IT Director.

The main tasks for this exercise are as follows:

1. View Database Settings
2. Amend Database Settings
3. View and Amend Server Settings

▶ Task 1: View Database Settings

1. Start the PowerShell ISE as Administrator.
2. Open **D:\Labfiles\Lab11\Starter\DisplayProperties.ps1**.

3. Select and run the code to import the SQL PowerShell module and test that it has been loaded by typing **Get-PSProvider** in the console window.
4. Select and run the code to set the location. Check that the directory has been changed correctly.
5. Select and run the code to assign the database object to the $database variable. Type **Write-Host $database.Name** to show the name of the object. To check it is an object, type **$database | Get-Member**.
6. Select and run the code to display some of the database properties.
7. Select and run the code to display some of the database options.

▶ Task 2: Amend Database Settings

1. Open **D:\Labfiles\Lab11\Starter\ChangeProperties.ps1**.
2. Select and run the code to import the SQL PowerShell module.
3. Select and run the code to set the location.
4. Select and run the code to assign the database object to the **$database** variable.
5. Select and run the code to change the compatibility level of the AdventureWorks2016 database.
6. Select and run the code to change some of the database options.

▶ Task 3: View and Amend Server Settings

1. Open **D:\Labfiles\Lab11\Starter\ChangeServerSettings.ps1**.
2. Select and run the code to import the SQL PowerShell module.
3. Select and run the code to set the location.
4. Select and run the code to assign the server object to the **$server** variable.
5. Select and run the code to view properties of the server object, and then to display the properties as a formatted list.
6. Select and run the code to change the login mode for the server.
7. Select and run the code to change the login mode back to integrated.
8. Select and run the code to view some server settings.
9. Select and run the code to change some server settings.
10. Select and run the code to reset some server settings back to their original values.
11. Close the PowerShell ISE.

Results: After completing this lab exercise, you will have PowerShell scripts to show the IT Director.

Check Your Knowledge

Question	
What is an SMO object?	
Select the correct answer.	
	A SQL PowerShell provider.
	An object with which part of SQL Server can be managed programmatically.
	A SQL Server feature that improves performance.
	Part of the Windows operating system.
	The top level object in the SQL Server hierarchy.

Question: Can you name three ways of getting information about a cmdlet?

Question: When would you use a PowerShell provider?

Module Review and Takeaways

PowerShell is an object-oriented scripting language with which you can automate SQL Server tasks. By accessing the SMO, you can view and amend properties, create new objects such as databases or tables, and automate administrative jobs.

PowerShell scripts can be run repeatedly, with predictable results, making SQL Server tasks faster and reducing the number of problems.

 **Best Practice:** Use aliases when working with the console window, but make scripts easy to read by using correctly capitalized cmdlet names.

It takes a little time to understand how PowerShell works, but the investment will pay off when tasks have to be run repeatedly, or when you have to be certain that tasks have been fully tested.

The efficiency improvements that can be gained by automating tasks are immense, and will be well worth the investment in learning to use PowerShell effectively.

Review Question(s)

Question: What tasks might benefit from automating with PowerShell for your SQL Server environment?

Module 12

Tracing Access to SQL Server with Extended Events

Contents:

Module Overview

Monitoring performance metrics provides a great way to assess the overall performance of a database solution. However, there are occasions when you need to perform more detailed analysis of the activity occurring within a Microsoft® SQL Server® instance—to troubleshoot problems and identify ways to optimize workload performance.

SQL Server Extended Events is a flexible, lightweight event-handling system built into the Microsoft SQL Server Database Engine. This module focuses on the architectural concepts, troubleshooting strategies and usage scenarios of Extended Events.

Note: Extended Events has been available in Microsoft Azure® SQL Database as a preview feature since October, 2015; at the time of publication, no date has been published for the General Availability (GA) of Extended Events in Azure SQL Database.

For more information about Extended Events in Azure SQL Database, see:

Extended Events in SQL Database

https://aka.ms/Rsp6gz

Objectives

After completing this module, you will be able to:

- Describe Extended Events core concepts.
- Create and query Extended Events sessions.

Lesson 1
Extended Events Core Concepts

This lesson focuses on the core concepts of Extended Events—the architectural design of the Extended Events engine, and core concepts of Extended Events are covered in depth.

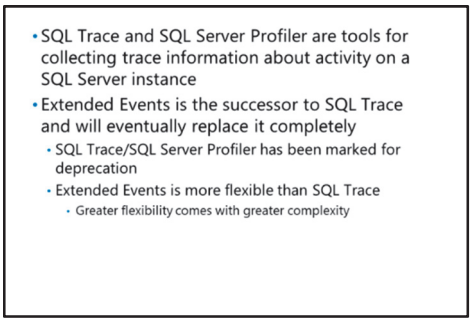
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the differences between SQL Server Profiler, SQL Trace, and Extended Events.
- Describe Extended Events architecture.
- Define Extended Events packages.
- Define Extended Events events.
- Define Extended Events predicates.
- Define Extended Events actions.
- Define Extended Events targets.
- Define Extended Events sessions.
- Define Extended Events types and maps.

Extended Events, SQL Trace, and SQL Server Profiler

Extended Events, SQL Trace, and SQL Server Profiler are all tools that you can use to monitor SQL Server events.



SQL Trace

SQL Trace is a server-side, event-driven activity monitoring tool; it can capture information about more than 150 event classes. Each event returns data in one or more columns and you can filter column values. You configure the range of events and event data columns in the trace definition; you can configure the destination for the trace data, a file or a database table, in the trace definition.

SQL Trace is included was SQL Server 7.0 and later versions.

SQL Server Profiler

SQL Server Profiler is a GUI for creating SQL traces and viewing data from them.

SQL Server Profiler was included in SQL Server 7.0 and later versions.

As established parts of the SQL Server platform, SQL Server Profiler and SQL Trace are familiar to many SQL Server administrators.

Note: SQL Trace and SQL Server Profiler have been marked for deprecation. SQL Server includes SQL Server Profiler and SQL Trace, but Microsoft intends to remove both tools in a future version of SQL Server. Extended Events is now the activity tracing tool recommended by Microsoft.

Extended Events

Extended Events was introduced in SQL Server 2008. Like SQL Trace, Extended Events is an event-driven activity monitoring tool; however, it attempts to address some of the limitations in the design of SQL Trace by following 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, which, in most cases, results in Extended Events having a lower performance overhead than an equivalent SQL Trace.

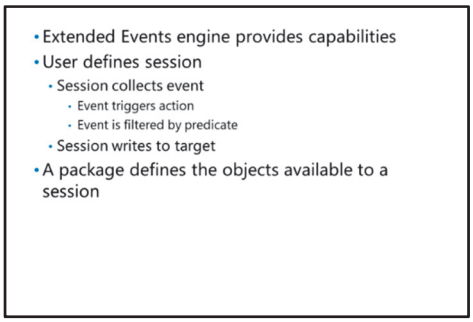
With Extended Events, you can define sophisticated filters on captured data. In addition to using value filters, you can filter events by sampling and data can be aggregated at the point it is captured. You can manage Extended Events either through a GUI in SQL Server Management Studio (SSMS) or by using Transact-SQL statements.

You can integrate Extended Events with the Event Tracing for Windows (ETW) framework, so that you can monitor SQL Server activity alongside other Windows® components.

The additional flexibility of Extended Events comes at the cost of greater complexity compared to the previous tools.

Extended Events Architecture

The Extended Events engine is a collection of services, running in the database engine, which provides the resources necessary for events to be defined and consumed.



As a user of Extended Events, you might find it most helpful to think about Extended Events primarily in terms of the session object. A session defines the Extended Events data that you want to collect, how the data will be filtered, and how the data will be stored for later analysis. Sessions are the top-level object through which you will interact with Extended Events:

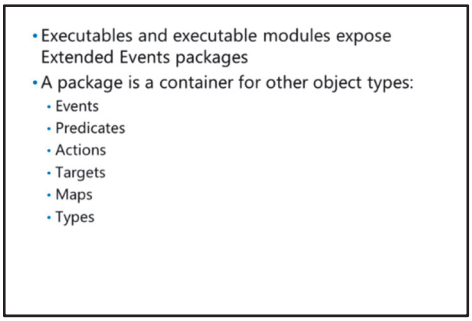
- User defines **session**
  - **Session** includes **event**
    - **Event** triggers **action**
    - **Event** is filtered by **predicate**
  - **Session** writes to **target**

A list of sessions is maintained by the Extended Events engine. You can define and modify sessions using Transact-SQL or in SSMS. You can view data collected by active sessions using Transact-SQL, in which case the data is presented in XML format, or by using SSMS.

Packages

Packages act as containers for the Extended Events objects and their definitions; a package can expose any of the following object types:

- Events
- Predicates
- Actions
- Targets
- Types
- Maps



Packages are contained in a module that exposes them to the Extended Events engine. A module can contain one or more packages, and can be compiled as an executable or DLL file.

A complete list of packages registered on the server can be viewed using the **sys.dm_xe_packages** DMV:

sys.dm_xe_packages

```
SELECT * FROM sys.dm_xe_packages;
```

For more information on **sys.dm_xe_packages**, see the topic *sys.dm_xe_packages (Transact-SQL)* in the SQL Server Technical Documentation:

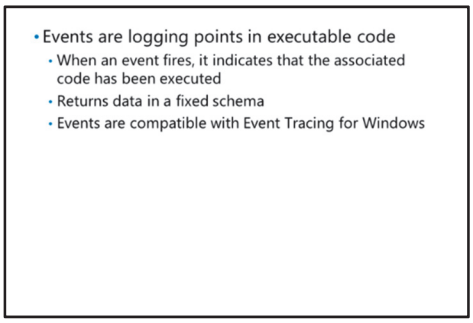
sys.dm_xe_packages (Transact-SQL)

https://aka.ms/Lonpcq

Events

Events are points in the code of a module that are of interest for logging purposes. When an event fires, it indicates that the corresponding point in the code was reached. Each event type returns information in a well-defined schema when it occurs.

All available events can be viewed in the **sys.dm_xe_objects** DMV under the **event** object_type:



sys.dm_xe_objects; events

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'event';
```

Events are defined by the Event Tracing for Window model—this means that SQL Server Extended Events can be integrated with ETW. Like ETW events, Extended Events are categorized by:

- **Channel**. The event channel identifies the target audience for an event. These channels are common to all ETW events:
  - **Admin**: events for administration and support.
  - **Operational**: events for problem investigation.
  - **Analytic**: high-volume events used in performance investigation.
  - **Debug**: ETW developer debugging events.
- **Keyword**. An application-specific categorization. In SQL Server, Extended Events event keywords map closely to the grouping of events in a SQL Trace definition.

A complete list of event keywords can be returned from **sys.dm_xe_map_values**:

Extended Events Event Keywords

```
SELECT map_value AS keyword
FROM sys.dm_xe_map_values
WHERE name = 'keyword_map'
ORDER BY keyword;
```

When you add, amend or remove an event from a package, you must refer to it with a two-part name: *package name.event name*.

A complete list of events and their package names can be returned by joining the list of events returned by the first example in this lesson to **sys.dm_xe_packages**:

Event and Package Names

```
SELECT xp.name AS package_name,
xo.name AS event_name,
xo.[description] AS event_description
FROM sys.dm_xe_objects AS xo
JOIN sys.dm_xe_packages AS xp
ON xp.guid = xo.package_guid
WHERE object_type = 'event'
ORDER BY package_name, event_name;
```

To find all the attribute columns associated with an event, you should join **sys.dm_xe_objects** to **sys.dm_xe_object_columns**:

Extended Events Object Columns

```
SELECT xoc.* FROM sys.dm_xe_objects AS xo
JOIN sys.dm_xe_object_columns AS xoc
ON xoc.object_package_guid = xo.package_guid
AND xoc.object_name = xo.name
WHERE xo.object_type = 'event';
```

For more information on **sys.dm_xe_objects**, see the topic *sys.dm_xe_objects (Transact-SQL)* in the SQL Server Technical Documentation:

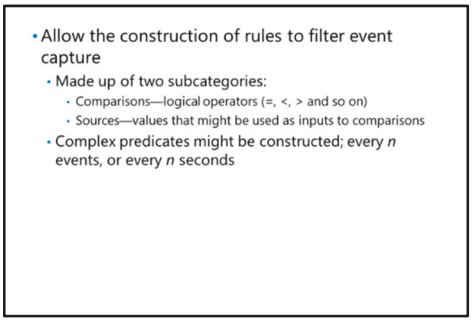
sys.dm_xe_objects (Transact-SQL)

https://aka.ms/Uhmmhw

Predicates

Predicates are logical rules with which events can be selectively captured, based on criteria you specify. Predicates divide into two subcategories:

- **Predicate comparisons**. Comparison operators, such as "equal to", "greater than", and "less than", which might make up a predicate filter. All predicate comparisons return a Boolean result (true or false).
- **Predicate sources**. Data items that might be used as inputs to predicate comparisons. These are similar to the column filters available when defining a SQL trace.



In addition to building logical rules, predicates are capable of storing data in a local context that means that predicates based on counters can be constructed; for example, every *n* events or every *n* seconds.

Predicates are applied to an event using a WHERE clause that functions like the WHERE clause in a Transact-SQL query.

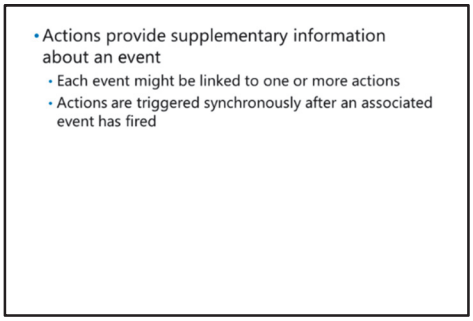
All available predicates can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** values **pred_source** and **pred_compare**:

sys.dm_xe_objects; predicates

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type LIKE 'pred%'
ORDER BY object_type, name;
```

Actions

Actions are responses to an event; you can use these responses to collect supplementary information about the context of an event at the time an event occurs. Each event may have a unique set of one or more actions associated with it. When an event occurs, any associated actions are raised synchronously.



Note: You might find the name of this object to be misleading; Extended Events actions do not allow you to define responses to an event. Instead, actions are additional steps that occur within the Extended Events engine when an event is triggered. Most actions provide more data to be collected about an event.

SQL Server defines more than 50 different actions, which include:

- **Collect database ID**
- **Collect T-SQL stack**
- **Collect session ID**
- **Collect session's NT username**
- **Collect client hostname**

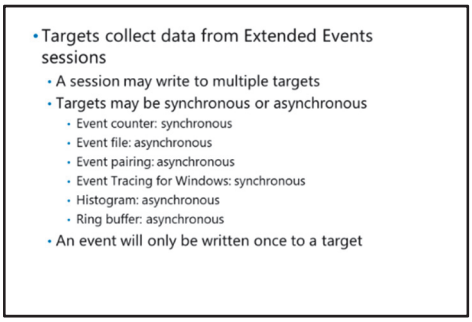
All available actions can be viewed in the DMV **sys.dm_xe_objects**, under the **object_type** value **action**:

sys.dm_xe_objects; actions

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'action';
```

Targets

Targets are the Extended Events objects that collect data. When an event is triggered, the associated data can be written to one or more targets. A target may be updated synchronously or asynchronously. The following targets are available for Extended Events:



- **Event counter**. The counter is incremented each time an event associated with a session occurs. Synchronous.
- **Event file**. Event data is written to a file on disk. Asynchronous.
- **Event pairing**. Tracks when events that normally occur in pairs (for example, lock acquired and lock released) do not have a matching pair. Asynchronous.
- **Event Tracing for Windows**. Event data is written to an ETW log. Synchronous.
- **Histogram**. A more complex counter that partitions counts by an event or action value. Asynchronous.
- **Ring buffer**. A first-in, first-out (FIFO) in-memory buffer of a fixed size. Asynchronous.

The design of Extended Events is such that an event will only be written once to a target, even if multiple sessions are configured to send that event to the same target.

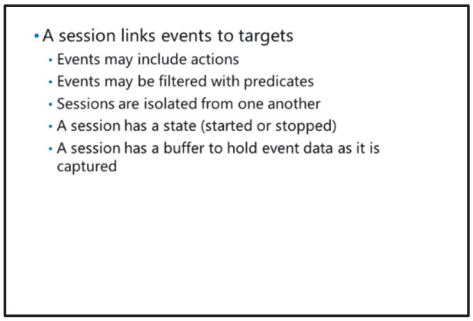
All available targets can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** value **target**:

sys.dm_xe_objects; targets

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'target';
```

Sessions

A session links one or more events to one or more targets. You can configure each event in a session to include one or more actions, and to be filtered with one or more predicates. Once defined, a session can be started or stopped as required; you can configure a session to start when the database engine starts.



A session may include events from more than one package. Sessions are isolated from one another; multiple sessions may use the same events and targets in different ways without interfering with one another.

A session is configured with a buffer in which event data is held while a session is running, before it is dispatched to the session targets. The size of this buffer is configurable, as is a dispatch policy (how long data will be held in the buffer). You can also configure whether or not to permit data loss from the buffer if event data arrives faster than it can be processed and dispatched to the session target.

All active Extended Events sessions can be viewed in the DMV **sys.dm_xe_sessions**:

sys.dm_xe_sessions

```
SELECT * FROM sys.dm_xe_sessions;
```

For more information on the set of DMVs for accessing information about active Extended Events sessions, including **sys.dm_xe_sessions**, see the topic *Extended Events Dynamic Management Views* in the SQL Server Technical Documentation:

Extended Events Dynamic Management Views

https://aka.ms/Jiinp1

All Extended Events sessions defined on a server can be returned by querying the DMV **sys.server_event_sessions**:

sys.server_event_sessions

```
SELECT * FROM sys.server_event_sessions;
```

For more information on the set of DMVs for accessing definitions for all Extended Events sessions, including **sys.server_event_sessions**, see the topic *Extended Events Catalog Views (Transact-SQL)* in the SQL Server Technical Documentation:

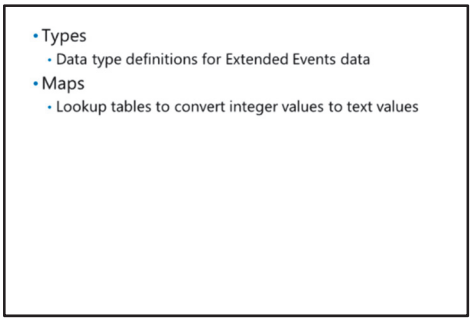
Extended Events Catalog Views (Transact-SQL):

https://aka.ms/Pwsolv

Note: A session can be created without targets, in which case the session data will only be visible through the **Watch Live Data** feature of SSMS.

Types and Maps

Types and maps are metadata objects that make it easier to work with Extended Events data. Types and maps are not directly referenced in an Extended Events session definition.



Types

Internally, Extended Events data is held in binary. A type identifies how a binary value should be interpreted and presented when the data is queried.

All available types can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** value **type**:

sys.dm_xe_objects; types

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'type';
```

Maps

A map is a lookup table for integer values. Internally, many event and action data values are stored as integers; maps link these integer values to text values that are easier to interpret.

All available types can be viewed in the DMV **sys.dm_xe_map_values**:

sys.dm_xe_map_values

```
SELECT * FROM sys.dm_xe_map_values
ORDER BY name, map_key;
```

Demonstration: Creating an Extended Events Session

In this demonstration, you will see how to create an Extended Events session.

Demonstration Steps

1. Ensure 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. In the **D:\Demofiles\Mod12** folder, run **Setup.cmd** as Administrator.
3. In the **User Account Control** dialog box, click **Yes** and wait for the script to finish.
4. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
5. On the **File** menu, point to **Open**, and then click **Project/Solution**.
6. In the **Open Project** dialog box, navigate to the **D:\Demofiles\Mod12** folder, click **Demo.ssmssln**, and then click **Open**.
7. In Solution Explorer, expand **Queries**, and then double-click **Demo 1 - create xe session.sql**.

8. Select code under the comment that begins **Step 1**, and then click **Execute** to create an Extended Events session.
9. Select code under the comment that begins **Step 2**, and then click **Execute** to verify that the session metadata is visible.
10. Select code under the comment that begins **Step 3**, and then click **Execute** to start the session and execute some queries.
11. Select code under the comment that begins **Step 4**, and then click **Execute** to query the session data.
12. Select code under the comment that begins **Step 5**, and then click **Execute** to refine the session data query.
13. In Object Explorer, under **MIA-SQL**, expand **Management**, expand **Extended Events**, expand **Sessions**, expand **SqlStatementCompleted**, and then double-click **package0.ring_buffer**.
14. In the **Data** column, click the XML value, and note that this is the same data that is returned by the query under the comment that begins **Step 4** (note that additional statements will have been captured because you ran the code earlier).
15. In Object Explorer, right-click **SqlStatementCompleted**, and then click **Watch Live Data**.
16. In the **Demo 1 - create xe sessions.sql** query pane, select the code under the comment that begins **Step 7**, and then click **Execute** to execute some SQL statements.
17. Return to the **MIA-SQL - SqlStatementCompleted: Live Data** pane. Wait for the events to be captured and displayed; this can take a few seconds. Other SQL statements from background processes might be captured by the session.
18. If the results do not appear, repeat steps 16 and 17.
19. In the **Demo 1 - create xe sessions.sql** query pane, select the code under the comment that begins **Step 8**, and then click **Execute** to stop the session.
20. In Object Explorer, right-click **SqlStatementCompleted**, and then click **Properties**.
21. In the **Session Properties** dialog box, review the settings on the **General**, **Events**, **Data Storage** and **Advanced** pages, if necessary referring back to the session definition under the comment that begins **Step 1**.
22. In the **Session Properties** dialog box, click **Cancel**.
23. Select the code under the comment that begins **Step 10**, and then click **Execute** to drop the session.
24. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which system DMV provides the list of events configured in an active Extended Events session?	
Select the correct answer.	
	sys.dm_xe_session _targets
	sys.dm_xe_session_events
	sys.dm_xe_sessions
	sys.dm_xe_session_event_actions

Lesson 2
Working with Extended Events

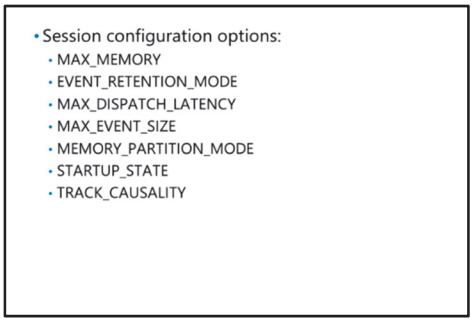
This lesson discusses using Extended Events. It covers common scenarios in which you might create Extended Events sessions for troubleshooting and performance optimization, as well as the system_health Extended Events session, which captures several events relevant to performance tuning.

Lesson Objectives

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

- Configure Extended Events sessions.
- Configure Extended Events targets.
- Explain the system_health Extended Events session.
- Describe usage scenarios for Extended Events.
- Describe best practices for using Extended Events.

Configuring Sessions



As you have learned, Extended Events sessions are composed from several other object types—primarily, events and targets. Sessions also have a number of configuration options that are set at session level:

- MAX_MEMORY. The amount of memory allocated to the session for use as event buffers, in kilobytes. The default value is 4 MB.
- EVENT_RETENTION_MODE. Specifies how the session will behave when the event buffers are full and further events occur:
  - ALLOW_SINGLE_EVENT_LOSS. An event can be dropped from the session if the buffers are full. A compromise between performance and data loss, this is the default value.
  - ALLOW_MULTIPLE_EVENT_LOSS. Full event buffers containing multiple events can be discarded. Minimal performance impact, but high data loss.
  - NO_EVENT_LOSS. Events are never discarded; tasks that trigger events must wait until event buffer space is available. Potential for severe performance impact, but no data loss.
- MAX_DISPATCH_LATENCY. The amount of time events will be held in event buffers before being dispatched to targets. Defaults to 30 seconds. You may set this value to INFINITE, in which case the buffer is only dispatched when it is full, or the session is stopped.
- MAX_EVENT_SIZE. For single events larger than the size of the buffers specified by MAX_MEMORY, use this setting. If a value is specified (in kilobytes or megabytes), it must be greater than MAX_MEMORY.
- MEMORY_PARTITION_MODE:
  - NONE. Memory is not partitioned. A single group of event buffers is created.

  - PER_NODE. A group of event buffers is created per NUMA node.
  - PER_CPU. A group of event buffers is created per CPU.
- STARTUP_STATE. When set to ON, the session will start when SQL Server starts. The default value is OFF.
- TRACK_CAUSALITY. When set to ON, an identifier is added to each event identifying the task that triggered the event. With this, you can determine whether one event is caused by another.

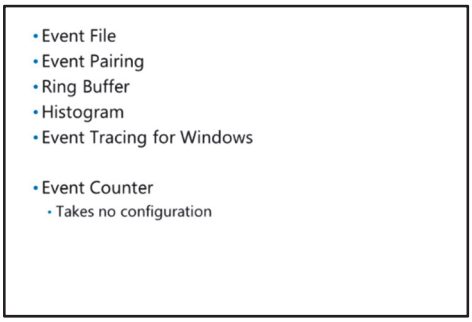
For more information about configuring a session through Transact-SQL, see the topic *CREATE EVENT SESSION (Transact-SQL)* in the SQL Server Technical Documentation:

CREATE EVENT SESSION (Transact-SQL)

https://aka.ms/Hhkkjl

Configuring Targets

Several Extended Events targets take configuration values when they are added to a session.



Event File

The Event File target can be used to write session data to a file. It takes the following configuration parameters:

- **filename**. The file name to write to; this can be any valid file name. If a full path is not specified, the file will be created in the \MSSQL\Log folder of the SQL Server instance on which the session is created.
- **max_file_size**. The largest size that the file may grow to; the default value is 1 GB.
- **max_rollover_files**. The number of files that have reached max_file_size to retain. The oldest file is deleted when this number of files is reached.
- **increment**. The file growth increment, in megabytes. The default value is twice the size of the session buffer.

For more information on configuring the event file target, see the topic *Event File Target* in the SQL Server Technical Documentation:

Event File Target

http://aka.ms/ixau4l

Event Pairing

The Event Pairing target is used to match events that occur in pairs (for example, statement starting and statement completing, or lock acquired and lock released), and report on beginning events that have no matching end event. It takes the following configuration parameters:

- **begin_event**. The beginning event name of the pair.
- **end_event**. The end event name of the pair.
- **begin_matching_columns**. The beginning event columns to use to identify pairs.

- **end_matching_columns**. The ending event columns to use to identify pairs.
- **begin_matching_actions**. The beginning event actions to use to identify pairs.
- **end_matching_actions**. The ending event actions to use to identify pairs.
- **respond_to_memory_pressure**. Permit the target to discard events (and so reduce memory consumption) when memory is under pressure.
- **max_orphans**. The maximum number of unpaired events the target will collect. The default value is 10,000. When this number is reached, events in the target are discarded on a first-in, first-out basis.

For more information on configuring the event pairing target, see the topic *Event Pairing Target* in the SQL Server Technical Documentation:

Event Pairing Target

http://aka.ms/lj7gng

Ring Buffer

The Ring Buffer target is used to write session data into a block of memory. When the allocated memory is full, the oldest data in the buffer is discarded and new data is written in its place. The ring buffer target takes the following configuration parameters:

- **max_memory**. The maximum amount of memory the ring buffer might use, in kilobytes.
- **max_event_limit**. The maximum number of events the ring buffer might hold. The default value is 1,000.
- **occurrence_number**. The number of events of each type to keep in the ring buffer. When events are discarded from the buffer, this number of each event type will be preserved. The default value, zero, means that events are discarded on a pure first-in, first-out basis.

Events are dropped from the buffer when either the max_memory or max_event_limit value is reached.

For more information on configuring the ring buffer target, see the topic *Ring Buffer Target* in the SQL Server Technical Documentation:

Ring Buffer Target

http://aka.ms/y3c84h

Histogram

The Histogram target is used to partition a count of events into groups based on a specified value. It takes the following configuration parameters:

- **slots**. The maximum number of groups to retain. When this number of groups is reached, new values are ignored. Optional.
- **filtering_event_name**. The event that will be counted into groups. Optional. If not supplied, all events are counted.
- **source_type**. The type of object used for grouping. 0 for an event; 1 for an action.
- **source**. The event column or action that is used to create group names and partition the count of events.

For more information on configuring the histogram target, see the topic *Histogram Target* in the SQL Server Technical Documentation:

Histogram Target

http://aka.ms/j9qkw9

Event Tracing for Windows

The Event Tracing for Windows target is used to write session data to an ETW log. It takes the following configuration parameters:

- **default_xe_session_name**. The name for the ETW session. There can only be one ETW session on a machine, and it will be shared between all instances of SQL Server. The default value is XE_DEFAULT_ETW_SESSION.
- **default_etw_session_logfile_path**. The path for the ETW log file. The default value is %TEMP%\XEEtw.etl.
- **default_etw_session_logfile_size_mb**. The log file size, in megabytes. The default value is 20 MB.
- **default_etw_session_buffer_size_kb**. The event buffer size. The default value is 128 KB.
- **retries**. The number of times to retry publishing to ETW before discarding the event. The default is 0.

For more information on configuring the Event Tracing for Windows target, see the topic *Event Tracing for Windows Target* in the SQL Server Technical Documentation:

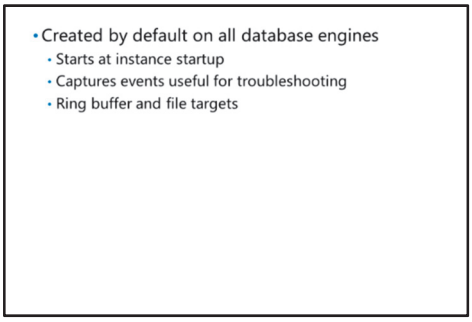
Event Tracing for Windows Target

https://aka.ms/Uml3t8

Event Counter

The Event Counter target is used to count events in a session. It takes no configuration parameters.

The system_health Extended Events Session



The system_health Extended Events session is created by default when a database engine instance is installed. The session is configured to start automatically when the database engine starts. The system_health session is configured to capture a range of events that are relevant for troubleshooting common SQL Server issues. In SQL Server, these include:

- Details of deadlocks that are detected, including a deadlock graph.
- The **sql_text** and **session_id** when an error that has a severity of 20 (or higher) occurs.
- The **sql_text** and **session_id** for sessions that encounter a memory-related error.
- The **callstack**, **sql_text**, and **session_id** for sessions that have waited for more than 15 seconds on selected resources (including latches).

- The **callstack**, **sql_text**, and **session_id** for any sessions that have waited for 30 seconds or more for locks.
- The **callstack**, **sql_text**, and **session_id** for any sessions that have waited for a long time for preemptive waits. (A preemptive wait occurs when SQL Server is waiting for external API calls to complete. The trigger time varies by wait type).
- The **callstack** and **session_id** for CLR allocation and virtual allocation failures (when insufficient memory is available).
- A record of any nonyielding scheduler problems.
- The **ring_buffer** events for the memory broker, scheduler monitor, memory node OOM, security, and connectivity. This tracks when an event is added to any of these ring buffers.
- System component results from **sp_server_diagnostics**.
- Instance health collected by **scheduler_monitor_system_health_ring_buffer_recorded**.
- Connectivity errors using **connectivity_ring_buffer_recorded**.
- Security errors using **security_error_ring_buffer_recorded**.

The system_health session writes data to two targets:

- A ring buffer target, configured to hold up to 5,000 events and to occupy no more than 4 MB.
- An event file target, composed of up to four files of 5 MB each.

 **Note:** The details of the system_health session are best understood by looking at its definition. You can generate a definition from SSMS:

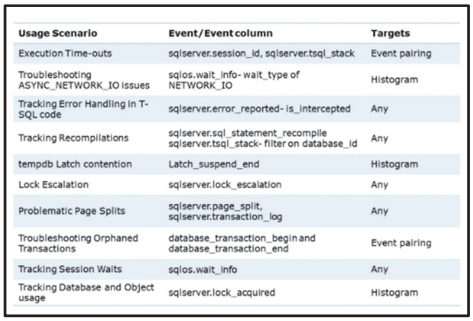
1. Connect SSMS Object Explorer to any SQL Server instance on which you have administrative rights.
2. In the Object Explorer pane, expand **Management**, expand **Extended Events**, and then expand **Sessions**.

Right-click the **system_health** node, click **Script As**, click **CREATE TO**, and then click **New Query Editor Window**. A script to recreate the system_health session will be generated.

Because both targets are configured to roll over and discard the oldest data they contain when they are full, the system_health session will only contain the most recent issues. On instances of SQL Server where the system_health session is capturing a lot of events, the targets may roll over before you can examine specific events.

Usage Scenarios for Extended Events

Extended Events can be used to troubleshoot many common performance issues.



Usage Scenario	Event/Event column	Targets
Execution Time-outs	sqlserver.session_id, sqlserver.tsql_stack	Event pairing
Troubleshooting ASYNC_NETWORK_IO issues	sqlos.wait_info- wait_type of NETWORK_IO	Histogram
Tracking Error Handling in T-SQL code	sqlserver.error_reported- is_intercepted	Any
Tracking Recompilations	sqlserver.sql_statement_recompile sqlserver.tsql_stack- filter on database_id	Any
tempdb Latch contention	Latch_suspend_end	Histogram
Lock Escalation	sqlserver.lock_escalation	Any
Problematic Page Splits	sqlserver.page_split, sqlserver.transaction_log	Any
Troubleshooting Orphaned Transactions	database_transaction_begin and database_transaction_end	Event pairing
Tracking Session Waits	sqlos.wait_info	Any
Tracking Database and Object usage	sqlserver.lock_acquired	Histogram

Execution Time-outs

When a Transact-SQL statement runs for longer than the client application's command time-out setting, a time-out error will be raised by the client application. Without detailed client application logging, it might be difficult to identify the statement causing a time-out.

This scenario is an ideal use case for the Extended Events event pairing target, using either of the following pairs:

- **sqlserver.sp_statement_starting** and **sqlserver.sp_statement_completed** (for systems using stored procedures for database access).
- **sqlserver.sql_statement_starting** and **sqlserver.sql_statement_completed** (for systems using ad-hoc SQL for database access).

When a time-out occurs, the starting event will have no corresponding completed event, and will be returned in the output of the event pairing target.

Troubleshooting ASYNC_NETWORK_IO

The ASYNC_NETWORK_IO wait type occurs when the database engine is waiting for a client application to consume a result set. This can occur because the client application processes a result set row-by-row as it is returned from the database server.

To troubleshoot this issue with Extended Events, capture the **sqlos.wait_info** event, filtering on **wait_type** value NETWORK_IO. The histogram target might be suitable for this investigation, using either the client application name or the client host name to define histogram groups.

Tracking Errors and Error Handling in Transact-SQL

Errors may be handled in Transact-SQL code by using TRY…CATCH blocks. Every error raises an event in Extended Events; this includes the errors handled by the TRY…CATCH blocks. You might want to capture all unhandled errors, or track the most commonly occurring errors, whether or not they are handled.

The **sqlserver.error_reported** event can be used to track errors as they are raised. The **is_intercepted** column can be used to identify an error that is handled in a TRY…CATCH block.

Tracking Recompilations

Query execution plan recompilations occur when a plan in the plan cache is discarded and recompiled. High numbers of plan recompilations might indicate a performance problem, and may cause CPU pressure. Windows performance counters can be used to track overall recompilation counts for a SQL Server instance, but more detail might be needed to investigate further.

In Extended Events, the **sqlserver.sql_statement_recompile** event can provide detailed information, including the cause of recompilation.

The histogram target can be used for tracking recompilations. Group on **source_database_id** to identify the database with the highest number of recompilations in an instance. Group on **statement/object_id** to find the most commonly recompiled statements.

tempdb Latch Contention

Latch contention in **tempdb** can occur due to contention for the allocation bitmap pages when large numbers of temporary objects are being created or deleted. This causes **tempdb** performance problems because all allocations in **tempdb** are slowed down.

The **latch_suspend_end** event tracks the end of latch waits by **database_id**, **file_id**, and **page_id**. With the predicate **divide_evenly_by_int64**, you can capture the contention specifically on allocation pages, because the different allocation bitmap pages occur at regular intervals in a database data file. Grouping the events using the histogram target should make it easier to identify whether latch waits are caused by contention for allocation bitmap pages.

Tracking Lock Escalation

Lock escalation occurs when more than 5,000 locks are required in a single session or under certain memory conditions.

The **sqlserver.lock_escalation** event provides the lock escalation information.

Tracking Problematic Page Splits

Page splits are of two types:

- Mid-page splits.
- Page splits for new allocations.

Mid-page splits create fragmentation and more transaction log records due to data movement.

Tracking page splits alone, using the **sqlserver.page_split** event, is inefficient as it does not differentiate between the problematic mid-page splits and normal allocation splits. The **sqlserver.transaction_log** event can be used for tracking **LOP_DELETE_SPLIT** operation to identify the problematic page splits. A histogram target might be most suitable for this task, grouping either on **database_id** (to find the database with the most page splits) or, within a single database, on **alloc_unit_id** (to find the indexes with the most page splits).

Troubleshooting Orphaned Transactions

Orphaned transactions are open transactions where the transaction is neither committed nor rolled back. An orphaned transaction might hold locks and lead to more critical problems like log growth and blocking, potentially leading to a block on the whole SQL Server instance.

The **database_transaction_begin** and **database_transaction_end** events can be used with an event pairing target to identify the orphaned transactions. The **tsql_frame** action can be used to identify the line of code where the orphaned transaction started.

Tracking Session-level Wait Stats

The wait stats available from the **sys.dm_os_wait_stats** DMV are aggregated at instance level, so it's not a fine-grained troubleshooting tool. Although you can track wait stats by session with the additional **sys.dm_exec_session_wait_stats** DMV on SQL Server, this might not be suitable for use in a busy system with many concurrent database sessions.

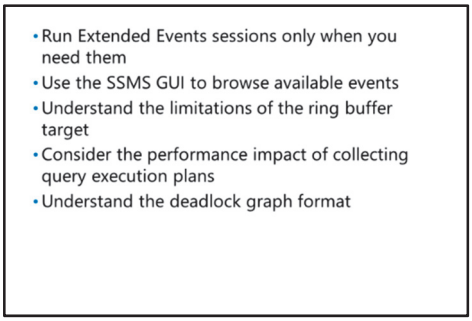
The **sqlos.wait_info** event can be used to track waits across multiple concurrent sessions.

Tracking Database and Object Usage

Tracking database and objects usage helps to identify the most frequently used database and most frequently used objects within a database. You might use this information to guide your optimization efforts, or to prioritize objects for migration to faster storage or memory-optimized tables.

The **sqlserver.lock_acquired** event can help with tracking the usage in most cases. For database usage, a histogram target grouping on **database_id**. Object usage can be tracked by tracking SCH_M or SCH_S locks at the object resource level by grouping on **object_id** in a histogram target.

Extended Events Best Practices



Run Extended Events Sessions Only When You Need Them

Although Extended Events is a lightweight logging framework, each active session has an overhead of CPU and memory resources. You should get into the practice of only running Extended Events sessions you have created when you have to troubleshoot specific issues.

Use the SSMS GUI to Browse Available Events

The Events page of the Extended Events GUI in SSMS brings all the metadata about individual events together into one view; this view helps you understand that the information that Extended Events makes available is easier than querying the DMVs directly.

Understand the Limitations of the Ring Buffer Target

When using a ring buffer target, you might not always be able to view all the events contained in the ring buffer. This is due to a limitation of the **sys.dm_xe_session_targets** DMV; the DMV is restricted to displaying 4 MB of formatted XML data. Because Extended Events data is stored internally as unformatted binary, it is possible that the data in a ring buffer will, when converted to formatted XML, exceed the 4 MB limit of **sys.dm_xe_session_targets**.

You can test for this effect by comparing the number of events returned from a ring buffer in XML with the count of events returned in the XML document header—or check the value of the **truncated** attribute in the XML header. In this example, the query is comparing these values for the system_health session:

Ring Buffer: Number of Events in XML Compared with Header

```
SELECT x.target_data.value('(RingBufferTarget/@eventCount)[1]', 'int') AS event_count,
             x.target_data.value('count(RingBufferTarget/event)', 'int') AS node_count,
             x.target_data.value('(RingBufferTarget/@truncated)[1]', 'bit') AS
output_truncated
FROM   (      SELECT CAST(target_data AS xml) AS target_data
                    FROM sys.dm_xe_sessions AS xs
                    JOIN sys.dm_xe_session_targets AS xst
                    ON xs.address = xst.event_session_address
                    WHERE xs.name = N'system_health'
                    AND xst.target_name = N'ring_buffer' ) AS x
```

To avoid this effect, you can:

- Use a file-based target (event file target or ETW target).
- Reduce the size of the MAX_MEMORY setting for the ring buffer to reduce the likelihood that the formatted data will exceed 4 MB. No single value is guaranteed to work; you might have to try a setting, and be prepared to adjust it, to minimize the truncation effect while still collecting a useful volume of data in the ring buffer.

 **Note:** This effect is not strictly limited to the ring buffer target; it can occur on any target that stores output in memory buffers (ring buffer target, histogram target, event pairing target, and event counter target), but it is most likely to affect the ring buffer target because it stores unaggregated raw data. All the other targets using memory buffers contain aggregated data, and are therefore less likely to exceed 4 MB when formatted.

Consider the Performance Impact of Collecting Query Execution Plans

Three events can be used to collect query execution plans as part of an Extended Events session:

- **query_post_compilation_showplan**. Returns the estimated query execution plan when a query is compiled.
- **query_pre_execution_showplan**. Returns the estimated query execution plan when a query is executed.
- **query_post_execution_showplan**. Returns the actual query execution plan when a query is executed.

When using any of these events you should consider that adding them to a session, even when predicates are used to limit the events captured, can have a significant impact on the performance of the database engine instance. This effect is most marked with the **query_post_execution_showplan** event. You should limit your use of these events to troubleshooting specific issues; they should not be included in an Extended Events session that is always running.

Understand the Deadlock Graph Format

Deadlock graphs collected by the **xml_deadlock_report** and **database_xml_deadlock_report** events are in a different format from the deadlock graphs produced by SQL Server Profiler; with these, you can use deadlock graphs captured by Extended Events to represent complex deadlock scenarios involving more than two processes. If saved as an .xdl file, both formats of deadlock graph can be opened by SSMS.

Demonstration: Tracking Session-Level Waits

In this demonstration, you will see how to report on wait types by session using Extended Events.

Demonstration Steps

1. In SSMS, in Solution Explorer, double-click **Demo 2 - track waits by session.sql**.
2. In Object Explorer, expand **Management**, expand **Extended Events**, right-click **Sessions**, and then click **New Session**.
3. In the **New Session** dialog box, on the **General** page, in the **Session name** box, type **Waits by Session**.
4. On the **Events** page, in the **Event library** box, type **wait**, and then, in the list below, double-click **wait_info**, to add it to the **Selected events** list.
5. Click **Configure** to display the **Event configuration options** list.
6. In the **Event configuration options** list, on the **Global Fields (Actions)** tab, select **session_id**.
7. On the **Filter (Predicate)** tab, click **Click here to add a clause**.
8. In the **Field** list, click **sqlserver.session_id**, in the **Operator** list, click **>**, and then in the **Value** box, type **50**. This filter will exclude most system sessions from the session.
9. On the **Data Storage** page, click **Click here to add a target**.

10. In the **Type** list, click **event_file**, in the **File name on server** box, type **D:\Demofiles\Mod12\waitbysession**, in the first **Maximum file size** box, type **5**, in the second **Maximum file size** box, click **MB**, and then click **OK**.
11. In Object Explorer, expand **Sessions**, right-click **Waits by Session**, and then click **Start Session**.
12. In File Explorer, in the **D:\Demofiles\Mod12** folder, right-click **start_load_1.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press Enter. Leave the workload to run for a minute or so before proceeding.
13. In SSMS, in the **Demo 2 - track waits by session.sql** pane, select the code under the comment that begins **Step 14**, click **Execute**, and then review the results.
14. Select the code under the comment that begins **Step 15**, and then click **Execute** to stop and drop the session, and to stop the workload.
15. In File Explorer, in the **D:\Demofiles\Mod12** folder, note that one (or more) files with a name matching **waitbysession*.xel** have been created.
16. Close File Explorer, close SSMS without saving changes, and then in the Windows PowerShell window, press Enter to close the window.

Categorize Activity

Place each Extended Events target type into the appropriate category. Indicate your answer by writing the category number to the right of each item.

Items	
1	Ring buffer target
2	Event file target
3	Histogram target
4	Event tracking for Windows target
5	Event pairing target
6	Event counter target

Category 1		Category 2
Written to Memory Buffers		Written to File on Disk

Lab: Extended Events

Scenario

While investigating the general slowness of the new SQL Server instance, you notice deadlock occurrences and excessive fragmentation in indexes caused by page split operations. In this lab, you will review deadlock occurrences using the default session, and implement a new Extended Event session to identify workloads that cause huge page splits.

Objectives

After completing this lab, you will be able to:

- Access data captured by the System Health Extended Events session.
- Create a custom Extended Events session.

Estimated Time: 90 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using the system_health Extended Events Session

Scenario

While investigating the general slowness of the new SQL Server instance, you are informed that users frequently report deadlock error messages in application logs. In this exercise, you will review the Extended Events default system_health session and analyze the output of deadlock event.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Run a Workload

3. Query the system_health Extended Events Session

4. Extract Deadlock Data

▶ Task 1: Prepare the Lab Environment

1. Ensure 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:\Labfiles\Lab12\Starter** folder as Administrator.

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab12\Starter** folder, run **start_load_1.ps1** with Windows PowerShell. If a message is displayed asking you to confirm a change in execution policy, type **Y**.

2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the system_health Extended Events Session

1. Start SQL Server Management Studio and connect to the **MIA-SQL** instance, then **D:\Labfiles\Lab12\Starter\Project\Project.ssmssln** and **Exercise 01 - system_health.sql**.
2. Under the comment that begins **Task 2**, edit and execute the query to return data from the system_health session, using the **sys.fn_xe_file_target_read_file** DMF to extract data from the session's event file target.

   Hint: you can examine the definition of the system_health session to find the file name used by the event file target.

▶ Task 4: Extract Deadlock Data

1. Under the comment that begins **Task 3**, edit and execute the query to extract the events with a name attribute of **xml_deadlock_report** from the XML version of the **event_data** column. Include the event time and the **/event/data/value/deadlock** element in your output.
2. Click any of the row values in the **deadlock_data** column to view the deadlock XML in detail.

Results: After completing this exercise, you will have extracted deadlock data from the SQL Server.

Exercise 2: Tracking Page Splits Using Extended Events

Scenario

While investigating the general slowness of the new SQL Server instance, you notice excessive fragmentation in indexes caused by page split operations. In this exercise, you will implement Extended Events to identify those workloads that cause huge page splits.

 **Note:** Although a **page_split** event is available, it doesn't provide enough information for you to discriminate between expected page splits (which occur when a table or index is extended, referred to as *end page splits*) and page splits that can harm performance (which occur when data must be inserted in the middle of a page, referred to as *mid-page splits*). You can detect mid-page splits by analyzing the **transaction_log** event.

The main tasks for this exercise are as follows:

1. Create an Extended Events Session to Track Page Splits
2. Run a Workload
3. Query the Session
4. Extract alloc_unit_id and Count Values
5. Return Object Names
6. Delete the Session

▶ Task 1: Create an Extended Events Session to Track Page Splits

1. In Solution Explorer, open **Exercise 02 - page splits.sql**.
2. Create a new Extended Events session on the **MIA-SQL** instance with the following properties:
   - Session name: **track page splits**
   - Event(s) included: **sqlserver.transaction_log**

- Event filter(s):
  - **operation** = **LOP_DELETE_SPLIT**
  - **database_name** = **AdventureWorks**
- Session target: **Histogram**
  - Filtering target: **sqlserver.transaction_log**
  - Source: **alloc_unit_id**
  - Source type: **event**

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab12\Starter** folder, execute **start_load_2.ps1** with PowerShell.
2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the Session

- In SSMS, in the query window for **Exercise 02 - page splits.sql,** under the comment that begins **Task 3**, edit and execute the query to extract data from the histogram target of the **track page splits** session. Use the **sys.dm_xe_session_targets** DMV to extract data from the session's histogram target. For this task, include only the **target_data** column in your output result set and cast the results to XML.

▶ Task 4: Extract alloc_unit_id and Count Values

- Under the comment that begins **Task 4**, edit and execute the query so that it returns the **count** attribute for each **HistogramTarget/Slot** node, and the **value** child node for each **HistogramTarget/Slot** node.

▶ Task 5: Return Object Names

- Under the comment that begins **Task 5**, edit and execute the query to join to **sys.allocation_units**, **sys.partitions** and **sys.indexes** to find the names of objects affected by page splits.

▶ Task 6: Delete the Session

1. Delete the **track page splits** session.
2. Close any open applications and windows.

Results: After completing this exercise, you will have extracted page split data from SQL Server.

Question: If an Extended Events session has no targets defined, how would you view the data generated by the session?

Module Review and Takeaways

In this module, you have learned about the Extended Events system and its components. You have learned about the objects that comprise an Extended Events session, and how to query the metadata about each object type to understand the information it provides.

You have learned how to create, amend and drop Extended Events sessions using either SSMS or Transact-SQL, in addition to learning about various methods for extracting data from session targets.

Review Question(s)

Check Your Knowledge

Question	
Which of the following sources does not contain detailed information about Extended Events event definitions?	
Select the correct answer.	
	SQL Server Management Studio Extended Events GUI.
	The DMV sys.dm_xe_objects.
	The SQL Server Technical Documentation.

Module 13

Monitoring SQL Server

Contents:

Module Overview

The Microsoft® SQL Server® Database Engine can run for long periods without the need for administrative attention. However, if you regularly monitor the activity that occurs on the database server, you can deal with potential issues before they arise.

SQL Server provides a number of tools that you can use to monitor current activity and record details of previous activity. You need to become familiar with what each of the tools does and how to use them.

It is easy to become overwhelmed by the volume of output that monitoring tools can provide, so you also need to learn techniques for analyzing their output.

Objectives

After completing this module, you will be able to:

- Monitor current activity.
- Capture and manage performance data.
- Analyze collected performance data.

Lesson 1
Monitoring Activity

Dynamic management objects (DMOs) provide insights directly into the inner operations of the SQL Server Database Engine and are useful for monitoring. SQL Server database administrators must become familiar with some of the more useful DMOs as part of a process of ongoing server monitoring.

SQL Server Management Studio provides Activity Monitor, which you can use to investigate both current issues such as: "Is one process being blocked by another process?" and recent historical issues such as: "Which query has taken the most resources since the server was last restarted?" You should become familiar with the capabilities of Activity Monitor.

The SQL Server processes also expose a set of performance-related objects and counters to Windows® Performance Monitor. These objects and counters enable you to monitor SQL Server as a part of monitoring the entire server.

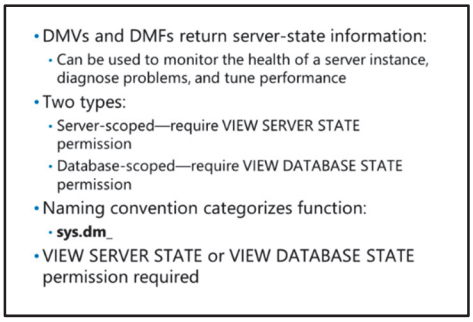
Lesson Objectives

After completing this lesson, you will be able to:

- Explain DMOs.
- View activity by using DMOs.
- Work with Activity Monitor in SQL Server Management Studio.
- Work with Performance Monitor.
- Work with SQL Server counters.

Overview of Dynamic Management Views and Functions

DMOs provide insight into the inner operation of the database engine. Some DMOs are used in the same way as views and are called dynamic management views (DMVs). Other DMOs are used in the same way as functions and are called dynamic management functions (DMFs). You can query a DMV by using Transact-SQL in the same way that you can query other views. A DMF is a table-valued function that accepts one or more parameters.



Note: The information that DMOs expose is generally not persisted in the database. DMOs are virtual objects that return state information; the state is cleared when the server instance is restarted or an administrator clears the state information.

There are more than 150 DMOs covering many categories; DMOs return server-state information that you can use to monitor the health of a server instance, diagnose problems, and tune performance. There are two types of DMVs and DMFs:

- Server-scoped
- Database-scoped

 **Note:** Most database-scoped DMOs are available in Azure® SQL Database. Server-scoped DMOs are typically not available in Azure SQL Database, because it is a database-level service. The SQL Server Technical Documentation indicates the availability of a DMO on different SQL Server service types.

All DMVs and DMFs exist in the sys schema and follow the naming convention **dm_%**. They are defined in the hidden resource database and are mapped to other databases. The DMOs are organized into categories by a naming convention, as shown in the following table:

Category	Description
sys.dm_exec_%	These objects provide information about connections, sessions, requests, and query execution. For example, the **sys.dm_exec_sessions** DMV returns one row for every session that is currently connected to the server.
sys.dm_os_%	These objects provide access to information that is related to the SQL Server operating system. For example, the **sys.dm_os_performance_counters** DMV provides access to SQL Server performance counters without the need to access them by using operating system tools.
sys.dm_tran_%	These objects provide access to transaction management. For example, the **sys.dm_os_tran_active_transactions** DMV returns details of currently active transactions.
sys.dm_io_%	These objects provide information about I/O processes. For example, the **sys.dm_io_virtual_file_stats** DMF returns details of I/O performance and statistics for each database file.
sys.dm_db_%	These objects provide database-scoped information. For example, the **sys.dm_db_index_usage_stats** DMV returns information about how each index in the database has been used.

Required Permissions

To query a DMO, you must have the SELECT permission on the object and the VIEW SERVER STATE or VIEW DATABASE STATE permission, depending on whether the object is server-scoped or database-scoped. This enables you to selectively restrict access to DMVs and DMFs for a user or logon. To control access for a user, first create the user in the **master** database (with any user name) and then deny the user the SELECT permission on the DMVs or DMFs that you do not want him or her to access. After this, the user cannot select from these DMVs and DMFs, regardless of the database context of the user, because the DENY permission within that database context is processed first.

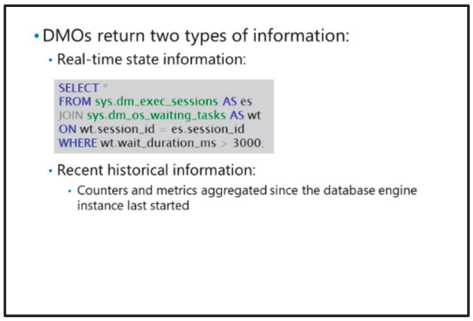
For more information about DMVs, see the topic *Dynamic Management Views and Functions (Transact-SQL)* in the SQL Server Technical Documentation:

 **Dynamic Management Views and Functions (Transact-SQL)**

https://aka.ms/Af2zcqLink

Viewing Activity by Using DMOs

In SQL Server Management Studio, you can see the list of available DMVs in Object Explorer, under the System Views node for any database. The DMFs are shown under the System Functions node, and then under the Table Valued Functions node of the **master** database.



Note: DMVs and DMFs need to be referenced in Transact-SQL statements with a two-part name using the sys schema as a prefix. They cannot be referenced by one-part names.

DMOs return two types of information:

- Real-time state information from the system.
- Recent historical information.

Real-Time State Information from the System

Most DMOs are designed to provide information about the current state of the system. In the example on the slide, two DMVs are being joined. The **sys.dm_exec_sessions** view returns one row for each current user session. The **sys.dm_os_waiting_tasks** view returns a row for each task that is currently waiting on a resource. By joining the two views and adding a filter, you can locate a list of user tasks that have been waiting for longer than 3,000 milliseconds (3 seconds). This type of query can be useful in indicating contention between sessions for server resources—such as I/O or memory—or database object resources—such as locks.

Historical Information

The second type of DMO returns historical information. This information typically takes the form of counters or metrics that are aggregated while the database engine instance is running. For example, you saw that the **sys.dm_os_waiting_tasks** view returned details of tasks that are currently waiting on resources. By comparison, the **sys.dm_os_wait_stats** view returns information about how often and how long any task had to wait for a specific **wait_type** since the SQL Server instance started.

Another useful example of a historical function is the **sys.dm_io_virtual_file_stats** DMF, which returns information about the performance of database files.

For more information about the **sys.dm_exec_sessions** DMV, see the topic *sys.dm_exec_sessions (Transact-SQL)* in the SQL Server Technical Documentation:

sys.dm_exec_sessions (Transact-SQL)

https://aka.ms/Rm8m7a

For more information about the **sys.dm_os_waiting_tasks** DMV, see the topic *sys.dm_os_waiting_tasks (Transact-SQL)* in the SQL Server Technical Documentation:

sys.dm_os_waiting_tasks (Transact-SQL)

https://aka.ms/Iicg1s

For more information about the **sys.dm_os_wait_stats** DMV, see the topic *sys.dm_os_wait_stats (Transact-SQL)* in the SQL Server Technical Documentation:

sys.dm_os_wait_stats (Transact-SQL)

https://aka.ms/Fzkyf1

For more information about the **sys.dm_io_virtual_file_stats** DMF, see the topic *sys.dm_io_virtual_file_stats (Transact-SQL)* in the SQL Server Technical Documentation:

sys.dm_io_virtual_file_stats (Transact-SQL)

https://aka.ms/Upoe0q

Demonstration: Using DMOs to View Activity

In this demonstration, you will see how to:

- Monitor activity on a database engine instance by using DMOs.

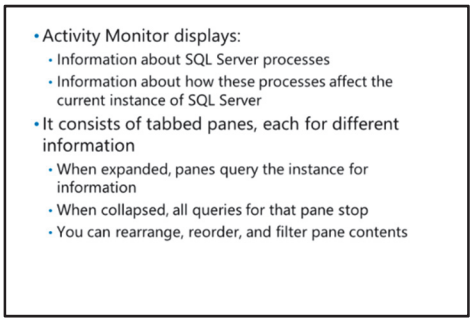
Demonstration Steps

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. In the **D:\Demofiles\Mod13** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to complete.
4. In the **D:\Demofiles\Mod13** folder, double-click **Workload1.cmd**.
5. Start **SQL Server Management Studio**, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
6. On the **file** menu, point to **Open**, click **Project/Solution**.
7. In the **Open Project** dialog box, navigate to the **D:\Demofiles\Mod13\Demo** folder, and then double-click **Demo.ssmssln**.
8. In Solution Explorer, expand **Queries**, and then double-click **Demo 1 - DMO.sql**.
9. Execute the code under the heading that begins with **Task 2** to view currently executing requests. Approximately 50 rows should be returned, but most are system requests.
10. Execute the code under the heading that begins with **Task 3** to view user processes.
11. Execute the code under the heading that begins with **Task 4** to filter executing requests by user sessions to show only user activity.
12. Execute the code under the heading that begins with **Task 5** to retrieve details of the Transact-SQL batch that is associated with each request.
13. Execute the code under the heading that begins with **Task 6** to show details of the Transact-SQL statement that is currently executing within each batch. This statement is complex, but it is fundamentally a substring operation on the results of the previous step.
14. Execute the code under the heading that begins with **Task 7** to stop the workload script. (The workload is configured to stop when the **##stopload** global temporary table is created.)

15. Execute the code under the heading that begins with **Task 8** to examine the contents of the query plan cache.
16. Execute the code under the heading that begins with **Task 9** to identify the top 10 most expensive queries in the query plan cache, based on average logical reads.
17. Execute the code under the heading that begins with **Task 10** to view I/O statistics for database files.
18. Execute the code under the heading that begins with **Task 11** to view wait statistics. The purpose of this query is to demonstrate the range of wait types that wait statistics are collected for.
19. On the **File** menu, click **Close**.
20. Leave SQL Server Management Studio open for the next demonstration.

Working with Activity Monitor in SQL Server Management Studio

Activity Monitor displays information about SQL Server processes and how these processes affect the current instance of SQL Server.



Benefits of Activity Monitor

Activity Monitor is a tabbed document window that contains the following expandable and collapsible panes:

- **Overview**. This section contains graphical information about processor usage, waiting tasks, database I/O, and batch requests per second.
- **Processes**. This section includes detailed information about processes, their IDs, logons, databases, and commands. This section will also show details of processes that are blocking other processes.
- **Resource Waits**. This section shows categories of processes that are waiting for resources and information about the wait times.
- **Data File I/O**. This section shows information about the physical database files in use and their recent performance.
- **Recent Expensive Queries**. This section shows detailed information about the most expensive recent queries and the resources that those queries consumed. You can right-click the queries in this section to view either the query or an execution plan for the query. When any pane is expanded, Activity Monitor queries the instance for information. When a pane is collapsed, all querying activity stops for that pane. You can expand more than one pane at the same time to view different kinds of activity on the instance.

You can customize the display of columns in Activity Monitor as follows:

- To rearrange the order of the columns, click the column heading, and then drag it to another location in the heading ribbon.
- To sort a column, click the column name.
- To filter on one or more columns, click the drop-down arrow in the column heading, and then select a value.

Note: The values that Activity Monitor exposes are also accessible by using DMOs. The tooltips for many of the column headers in Activity Monitor sections describe which DMO contains the corresponding information.

For more information about Activity Monitor, see the topic *Activity Monitor* in the SQL Server Technical Documentation:

Activity Monitor

https://aka.ms/Thoa2y

Demonstration: Using Activity Monitor in SQL Server Management Studio

In this demonstration, you will see how to use Activity Monitor to:

- View session activity.
- Resolve an instance of blocking.

Demonstration Steps

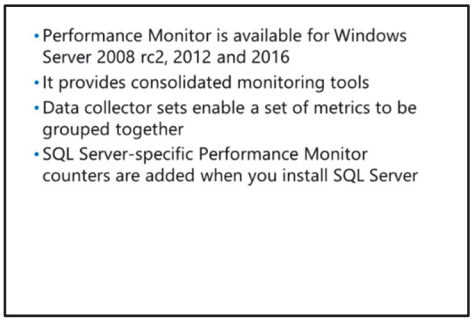
1. In SQL Server Management Studio, in Solution Explorer, double-click **Demo 2a - blocker.sql**.
2. Review the contents of the file and notice that it starts a transaction without committing it, and then click **Execute**.
3. In Solution Explorer, double-click **Demo 2b - blocked.sql**, and then click **Execute**. Notice that a result set is not returned and the query remains running; this query is blocked from accessing the **HR.Employees** table by the transaction that you opened in the previous step.
4. In Object Explorer, right-click **MIA-SQL**, and then click **Activity Monitor**.
5. In the **MIA-SQL Activity Monitor** pane, click **Processes** to expand the **Processes** section.
6. In the **Processes** section, in the **Database** column, in the column header, click the **Filter** button, and then click **InternetSales**.
7. Notice that one of the processes has a **Task State** of **SUSPENDED**; this is the query that is running in the Demo 2b - blocked.sql query pane.
8. Point to the column header for the **Blocked By** column to demonstrate that the column tooltip describes the DMO column that contains the information—the **sys.dm_os_waiting_tasks.blocking_session_id**.
9. In the **SUSPENDED** row, note the value in the **Blocked By** column. This is the session ID of the blocking session—the query in the Demo 2a - blocker.sql query pane.
10. In the **Processes** section, in the **Session ID** column, in the column header, click the **Filter** button, and then click the value of the session that you identified as the blocker in the previous step. Only one row should now be visible in the **Processes** section. Notice that the value in the **Head Blocker** column is **1**. This indicates that this session is the first in a blocking chain.
11. In the **Processes** section, right-click the row, and then click **Kill Process**.
12. In the **Kill Process** dialog box, click **Yes** to roll back the query in the Demo 2a - blocker.sql query pane.

Note: Note that processes should be killed only as a last resort.

13. Close the MIA-SQL Activity Monitor pane.
14. In the **Demo 2b - blocked.sql** query pane, notice that the query has completed because the block has been removed.
15. Close both panes.
16. In the **Microsoft SQL Server Management Studio** dialog box, click **No**.
17. Leave SQL Server Management Studio open for the next demonstration.

Working with Performance Monitor

Windows Performance Monitor is a tool that is available for Windows Server 2008 R2, 2012, and 2016. It brings together several performance and monitoring tools that are disparate in earlier versions of Windows Server.



Note: Performance Monitor is also available on the Windows Vista®, Windows 7, and Windows 10 desktop operating systems.

By consolidating several sources of information in one place, Windows Performance Monitor helps administrators to obtain the information they require to diagnose server performance and instability issues.

An important new feature in Windows Performance Monitor is the data collector set, which groups data collectors into reusable elements for use with different performance-monitoring scenarios. After a group of data collectors are stored as a data collector set, operations such as scheduling can be applied to the entire set by using a single property change. Windows Performance Monitor includes a default data collector set template so that system administrators can immediately begin to collect performance data that is specific to a server role or monitoring scenario.

Windows Performance Monitor is the key tool for monitoring Windows-based systems. SQL Server runs on the Windows operating system, so it is important to monitor at the server level in addition to the database engine level. This is because problems within the database engine might be caused by problems that occur outside at the operating system level.

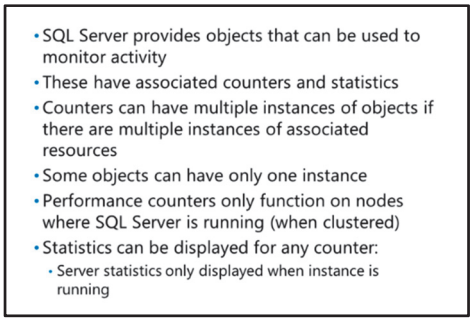
The main focus of the Windows Performance Monitor is on monitoring CPU, memory, disk system, and network. After you install SQL Server, a number of objects and counters that are related to SQL Server are available within Windows Performance Monitor.

For more information about Performance Monitor, see the topic *Windows Performance Monitor* on Technet:

Windows Performance Monitor

http://aka.ms/Faox3k

Working with SQL Server Counters



It is important to understand some of the basic terminology that is used within Windows Performance Monitor:

- The term object is used to represent a resource that can be monitored.
- Every object exposes one or more counters.
- A counter might have several instances if more than one resource of that type exists.

As an example, there is an object called **Processor** that consists of several counters. The **Processor** object provides metrics that are related to the processors that are available on the server. One of the commonly used counters for the **Processor** object is the **% Processor Time** counter. That counter then offers a set of instances that represent the individual processor cores that exist on the system. In addition, a value called **_Total** represents all the instances combined.

SQL Server provides objects and counters that Performance Monitor uses to monitor activity in computers that are running an instance of SQL Server. In this context, an object is any SQL Server resource, such as a SQL Server lock or Windows process. Each object contains one or more counters that determine various aspects of the objects to monitor. For example, the **SQLServer:Locks** object contains counters called **Number of Deadlocks/sec** and **Lock Timeouts/sec**.

The **SQLServer:Databases** object type has one instance for each database on SQL Server. Some object types (for example, the **SQLServer:Memory Manager** object) have only one instance. If an object type has multiple instances, you can add counters to track statistics for each instance, or in many cases, all instances at once. Counters for the default instance appear in the format **SQLServer:**<*object name*>. Counters for named instances appear in the format **MSSQL$**<*instance name*>:<*counter name*> or **SQLAgent$**<*instance name*>:<*counter name*>.

You can specify the SQL Server objects and counters that are monitored when Performance Monitor is started.

You can also configure Performance Monitor to display statistics from any SQL Server counter. In addition, you can set a threshold value for any SQL Server counter, and then generate an alert when a counter exceeds a predefined threshold.

Note: SQL Server statistics are displayed only when an instance of SQL Server is installed. If you stop and restart an instance of SQL Server, the display of statistics is interrupted and resumes automatically. Also note that you will see SQL Server counters in the Performance Monitor snap-in, even if SQL Server is not running. On a clustered instance, performance counters only function on the node where SQL Server is running.

SQL Server counters for Performance Monitor can be queried from Transact-SQL statements by using the **sys.dm_os_performance_counters** system DMV.

For more information about SQL Server objects and counters in Performance Monitor, see the topic *Use SQL Server Objects* in the SQL Server Technical Documentation:

Use SQL Server Objects

https://aka.ms/Lwwntb

For more information about the **sys.dm_os_performance_counters** system DMV, see the topic s*ys.dm_os_performance_counters (Transact-SQL)* in the SQL Server Technical Documentation:

sys.dm_os_performance_counters (Transact-SQL)

https://aka.ms/Dtkfv2

Demonstration: Using Performance Monitor

In this demonstration, you will see how to use:

- Default data collector sets.
- SQL Server Performance Monitor counters.

Demonstration Steps

1. Click **Start**, click **Administrative Tools**, and then double-click **Performance Monitor**.
2. In the **Performance Monitor** window, in the left pane, expand **Data Collector Sets**, expand **System**, and then, click **System Performance**.
3. In the right pane, double-click **Performance Counter**.
4. In the **Performance Counter Properties** dialog box, observe the different performance counters that this set collects, and then click **Cancel**.
5. In the left pane, right-click **System Performance**, and then click **Start**. Note that the symbol for the **System Performance** collector set changes to reflect that it is running. The collector set will collect data for one minute before it stops automatically. Wait for the collector set to finish, and the symbol to change back to its original form.
6. On the **Action** menu, click **Latest Report**. Notice that new nodes are added to the tree in the left pane. In the right pane, expand each section of the report to demonstrate the information that has been collected.
7. When you have finished reviewing the report, in the left pane, click **Performance Monitor**. Notice that the right pane changes to show a chart, with the **% Processor Time** counter preselected.
8. In the right pane, right-click anywhere in the pane, and then click **Add Counters**.
9. In the **Add Counters** dialog box, in the **Available counters** list, click the following counters, and then click **Add >>** to add them:
    a. **SQLServer:Memory Manager: Total Server Memory (KB)**
    b. **SQLServer:Memory Manager: Target Server Memory (KB)**
    c. **SQLServer:Databases: Percent Log Used** (Instance **InternetSales**)
    d. **MSSQL$SQL2:Memory Manager: Total Server Memory (KB)**
10. Click **OK**.

11. Review the changes to the Performance Monitor chart and then close Performance Monitor.
12. In SQL Server Management Studio, in Solution Explorer, double-click **Demo 3 - counters.sql**.
13. Execute the code in the file to demonstrate that SQL Server Performance Monitor counters are accessible by using the **sys.dm_os_performance_counters** system DMV.
14. On the **File** menu, click **Close**.
15. Leave SQL Server Management Studio open for the next demonstration.

    Question: Why might you use the **sys.dm_os_performance_counters** system DMV, instead of Performance Monitor, to access SQL Server counters?

Lesson 2
Capturing and Managing Performance Data

You have seen that DMOs provide useful information about the state of the system. The values that the DMOs provide are not generally persisted in any way and only reside in memory while the server is running. When the server instance is restarted, these values are reset.

When DMVs and DMFs were first introduced in SQL Server 2005, it was common for users to want to persist the values that the DMVs and DMFs provided. To this end, many users created a database to hold the values, and then created a job that would periodically collect and save the values.

The data collector system that was introduced with SQL Server 2008 formalizes this concept by creating a central warehouse for holding performance data, jobs for collecting and uploading the data to the warehouse, and a set of high quality reports that can be used to analyze the data. This lesson describes how to set up and configure the data collector. The next lesson describes the reports that are available from the data that the data collector has gathered.

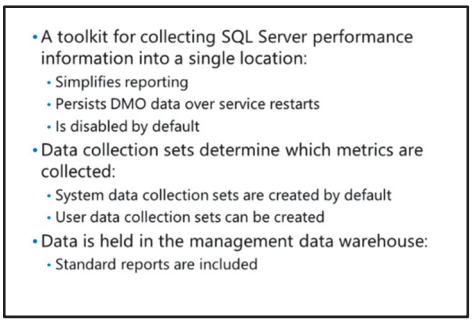
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the role of the data collector.
- Design a topology for use with the data collector.
- Configure the data collector.
- Configure security for the data collector.
- Monitor the data collector.

Overview of the Data Collector

Earlier in this module, you saw how you can use DMOs to obtain useful data. One disadvantage of using DMOs is that most views and functions only return real-time data, and the views and functions that return historic data are aggregations of the occurrences over time. However, to undertake effective performance tuning and monitoring, you will often require an overview over time, along with the ability to examine data in more detail when you investigate issues. The SQL Server data collector helps you to achieve this.



Data Collection Sets

When you configure the SQL Server data collector, a number of system data collection sets are created. These sets define the data that needs to be collected, how often the data is uploaded to a central repository, and how long the data is retained in that repository.

You can collect information from several locations. The data collector can:

- Query DMOs to retrieve detailed information about the operation of the system.

- Retrieve performance counters that provide metrics about the performance of both SQL Server and the entire server.
- Capture SQL Trace events that have occurred.

In addition to the system data collection sets, the SQL Server data collector can be extended by the creation of user-defined data collection sets. The ability to add user-defined data collection sets enables users to specify the data that they want to collect, and to use the infrastructure that the SQL Server data collector provides to collect and centralize the data.

To function, the SQL Server data collector depends upon a combination of SQL Server Agent jobs and SQL Server Integration Services (SSIS) packages. You will find the SQL Server data collector easier to work with if you are already familiar with these technologies.

Management Data Warehouse and Reports

The data that the SQL Server data collector gathers is stored in a central repository called a management data warehouse. The management data warehouse is a SQL Server database that you create by using a wizard that you can also use to configure data collection.

Three standard reports and a rich set of subreports are provided with the data collector, but you can write your own reports that are based on either the data that the system data collection sets gather, or data that user-defined data collection sets gather.

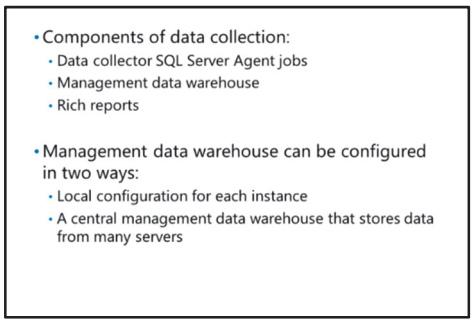
For more information about the SQL Server data collector, see the topic *Data Collection* in the SQL Server Technical Documentation:

Data Collection

https://aka.ms/Wy1dug

Designing a Data Collection Topology

The three components that make up the data collection system in SQL Server are:



- **The data collector**. This is made up of a set of SQL Server Agent jobs that run on the local server. They collect data from DMOs, performance counters, and SQL Trace events, and upload that data to a central repository.
- **The central repository**. This can consolidate data from multiple server instances.
- **The rich reports**. These are available to analyze the data in the management data warehouse. The reports are accessed by using SQL Server Management Studio.

A large enterprise should consider the use of a stand-alone system for the management data warehouse. There are two goals for creating a centralized management data warehouse:

- You can access reports that combine information for all server instances in your enterprise.
- You can offload the need to hold collected data, and to report on it from the production servers.

The data collector has two methods for uploading captured performance data into the central warehouse. Low volume information is sent immediately to the warehouse. Higher volume information is cached locally first, and then uploaded to the warehouse by using SSIS.

For more information about the management data warehouse, including a detailed description of the database schema, see the topic *Management Data Warehouse* in the SQL Server Technical Documentation:

Management Data Warehouse

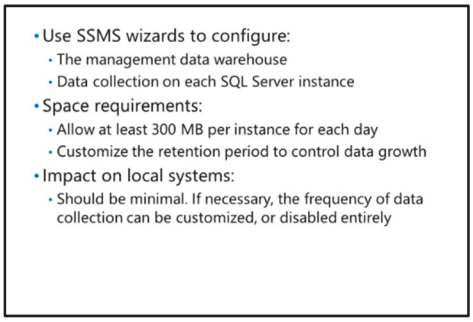
https://aka.ms/X6otzp

Configuring the Data Collector

Installing the data collection system in SQL Server involves two steps:

1. Configuring a central management data warehouse to hold the collected data.
2. Configuring each server instance to collect and upload the required data.

SQL Server provides a wizard that is used for both of these tasks. The first step when you run the wizard is to create a management data warehouse. One instance of the management data warehouse can store collection data from many SQL Server instances.



Planning Space Requirements

Sufficient disk space must be available to support the needs of the management data warehouse. The amount of disk space that is required will vary, based on the metrics you collect. However, in a typical configuration, you should allow at least 300 MB per day for each instance that loads data to the management data warehouse.

Impact on Local Systems

The only processes that are run on the local instances are the jobs that are used to collect and upload the data to the management data warehouse. Some data is collected very regularly, then cached locally and later uploaded by using SSIS. Other data is captured infrequently and uploaded immediately.

System data collection sets are created automatically during the setup of SQL Server data collection. You can enable and disable them as needed, and both the frequency of collection and the retention periods for collected data can be customized for system data collection sets and for user-defined data collection sets.

You must also consider the security requirements for the data collector. Security requirements are discussed in the next topic.

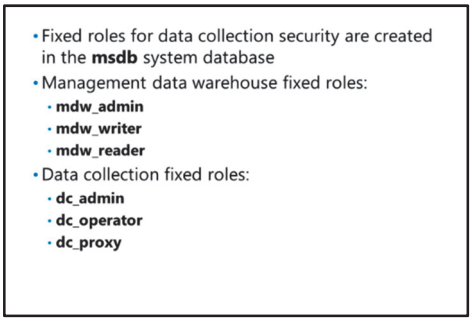
Note: If the SQL Server Agent has been configured to run as a System service account you may need to configure a proxy to upload the data into the management data warehouse.

For more information about how to configure SQL Server data collection, see the topic *Configure the Management Data Warehouse (SQL Server Management Studio)* in the SQL Server Technical Documentation:

Configure the Management Data Warehouse (SQL Server Management Studio)

https://aka.ms/Nkbr8d

Data Collection Security



The SQL Server data collector system has two security aspects that need to be considered:

- Who can access the management data warehouse?
- Who can configure data collection?

Accessing the Management Data Warehouse

Three fixed database roles are provided in the **msdb** system database to allow different levels of access to the management data warehouse:

- **mdw_admin**. Members of this role have full control of the management data warehouse, including permissions to read, write, update, and delete data from any table, alter the database schema, and perform administrative tasks on the management data warehouse.
- **mdw_writer**. Members of this role can write and upload data to the management data warehouse. All of the data collectors that write to an instance of the management data warehouse must be a member of this role.

Note: In addition to needing to be a member of the **mdw_writer** role to upload data, the jobs that collect the data need whatever permissions are required to access the data that they are collecting.

- **mdw_reader**. Members of this role can read data from the management data warehouse. A user needs to be a member of the **mdw_reader** role to be able to access the rich reports or directly read the collected data.

Configuring Data Collection

Three fixed database roles are provided in the **msdb** system database to allow different levels of access to the configuration of the data collector. Permissions for these roles are concentric; higher-privileged roles inherit the permissions of lower-privileged roles, in addition to their own permissions. For example, the **dc_admin** role inherits the permissions of the **dc_operator** and **dc_proxy** roles.

- **dc_admin**. Members of this role have full control of the data collection configuration, including:
  - Setting collector-level properties.
  - Adding new collection sets.
  - Installing new collection types.

 **Note:** Members of the **dc_admin** role must configure SQL Server Agent jobs, so the **dc_admin** role is a member of the **SQLServerAgentUser** role in the **msdb** system database.

For more information about the SQL Server Agent fixed roles, see the topic *SQL Server Agent Fixed Database Roles* in the SQL Server Technical Documentation:

 **SQL Server Agent Fixed Database Roles**

https://aka.ms/Nkbr8d

- **dc_operator**. Members of this role can read and update data collection configuration, including:
  - Starting or stopping a collection set.
  - Enumerating collection sets.
  - Changing the upload frequency for collection sets.
  - Changing the collection frequency for collection items that are part of a collection set.
- **dc_proxy**. Members of this role have read access to data collection configuration. They can also create SQL Server Agent job steps that use an existing proxy account, and execute SQL Server Agent jobs that they own.

 **Note:** Members of the **dc_operator** and **dc_proxy** roles must interact with SSIS packages, so both roles are a member of the **db_ssisltduser** and **db_ssisoperator** fixed roles in the **msdb** system database.

For more information about the SSIS fixed roles, see the topic *Integration Services Roles (SSIS Service)* in the SQL Server Technical Documentation:

 **Integration Services Roles (SSIS Service)**

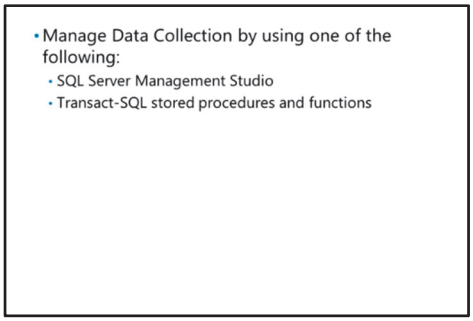
https://aka.ms/W82x28

For more information about data collection security, see the topic *Data Collector Security* in the SQL Server Technical Documentation:

 **Data Collector Security**

https://aka.ms/Xrnzt2

Monitoring Data Collector



As with other SQL Server Agent jobs, the history of the jobs that SQL Server data collector uses to capture and upload performance data is held in the SQL Server Agent job history tables, and can be viewed by using the standard SQL Server Agent job history viewer.

In addition to the standard job history, the data collector also logs configuration and other log information to tables in the **msdb** system database. The data collector calls stored procedures to add the log information, and also uses the SSIS logging features for the SSIS packages that it executes.

The data that is logged into the **msdb** system database is kept with the same retention period settings as the data collection sets that it relates to. The information that is retained can be viewed through the SQL Server Management Studio log file viewer or by querying the following objects in the **msdb** system database:

- **fn_syscollector_get_execution_details**. This table-valued function returns a portion of the SSIS log for data collection activity.
- **fn_syscollector_get_execution_stats**. This table-valued function returns detailed performance information about a data collection set or package.
- **syscollector_execution_log**. This view returns selected log information about a data collection set or package.
- **syscollector_execution_stats**. This view returns high-level statistics about a data collection set or package.

Three levels of logging are available and can be set by calling the **sp_syscollector_update_collection_set** system stored procedure. The lowest level of logging records starts and stops of collector activity. The next level of logging adds execution statistics and progress reports. The highest level of logging adds detailed SSIS package logging.

For more information about working with the data collector, see the topic *Manage Data Collection* in the SQL Server Technical Documentation:

Manage Data Collection

https://aka.ms/Xg801h

Demonstration: Configuring Data Collector

In this demonstration, you will see how to:

- Configure the management data warehouse.
- Enroll a data collector instance with a management data warehouse.
- Configure a system data collection set.

Demonstration Steps

Configure the Management Data Warehouse

1. In Object Explorer, under **MIA-SQL**, expand **Management**, right-click **Data Collection**, point to **Tasks**, and then click **Configure Management Data Warehouse**.
2. In the **Configure Management Data Warehouse Wizard** window, click **Next**.
3. On the **Configure Management Data Warehouse Storage** page, click **New**.
4. In the **New Database** dialog box, in the **Database name** box, type **MDW**, and then click **OK**.
5. On the **Configure Management Data Warehouse Storage** page, click **Next**.
6. On the **Map Logins and Users** page, review the available options, and then click **Next**.
7. On the **Complete the Wizard** page, click **Finish**.
8. Wait for the configuration process to complete, and then click **Close**.

Enroll the **MIA-SQL** Instance for Data Collection

1. In Object Explorer, under **Management**, right-click **Data Collection**, point to **Tasks**, and then click **Configure Data Collection**.
2. In the **Configure Data Collection Wizard** window, click **Next**.
3. On the **Setup Data Collection Sets** page, next to the **Server name** box, click the **Ellipsis** (**...**) button.
4. In the **Connect to Server** dialog box, verify that the **Server name** box has the value **MIA-SQL**, and then click **Connect**.
5. On the **Setup Data Collection Sets** page, in the **Database name** list, click **MDW**.
6. Under **Select data collector sets you want to enable**, select the **System Data Collection Sets** check box, and then click **Next**.
7. On the **Complete the Wizard** page, click **Finish**.
8. Wait for the configuration process to complete, and then click **Close**.

Configure a Data Collection Set

1. In Object Explorer, under **Management**, expand **Data Collection**, and then expand **System Data Collection Sets**. Observe that four collection sets are available but one of them is stopped.
2. Right-click **Disk Usage**, and then click **Properties**.
3. In the **Data Collection Set Properties** dialog box, on the **General** page, click **Pick**.
4. In the **Pick Schedule for Job** dialog box, click **CollectorSchedule_Every_5min** (the row with **ID** = **2**), and then click **OK**.
5. In the **Data Collection Set Properties** dialog box, click **OK**.
6. In Solution Explorer, under Queries, double click **SSRS Fix 2017.sql**.

7. Click **Execute**.
8. In File Explorer, in the **D:\Demofiles\Mod13** folder, start **Workload1.cmd** and allow it to run. This script will generate some activity that will appear in the data collection reports in the demonstrations in the next lesson.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? You can use the SQL Server data collector for real-time monitoring.	

Lesson 3
Analyzing Collected Performance Data

After performance data has been collected from a number of server instances and consolidated into a management data warehouse, the data can then be analyzed. You can write your own custom reports by using SQL Server Reporting Services, either by using custom reports in SQL Server Management Studio or by using Transact-SQL queries. Most users will find the standard reports that are supplied with SQL Server to be sufficient, without the need to write additional reports. You must be familiar with the information that is contained in the standard reports, and you must know how to navigate within the reports.

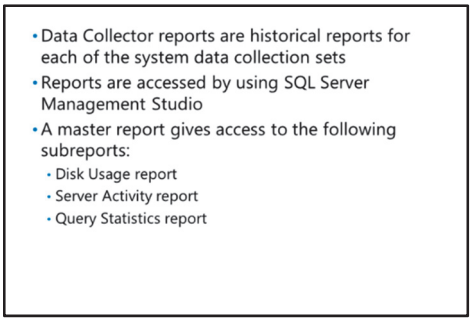
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the data collector reports.
- Use Disk Usage reports.
- Use Server Activity reports.
- Use Query Statistics reports.

Overview of Data Collector Reports

The SQL Server data collector provides a series of standard reports. These reports are based on the data that is collected from the system data collection sets—that has been consolidated into a centralized management data warehouse. The reports are accessed from within SQL Server Management Studio.



The following reports are available from the Management Data Warehouse Overview report that you start from SQL Server Management Studio:

- Disk Usage report
- Server Activity report
- Query Statistics report

Each report corresponds to one of the system data collection sets.

You can use these reports to obtain information for monitoring system capacity and troubleshooting system performance.

Note: Each report can be printed or exported to PDF or Microsoft Excel® files for further analysis.

For more information about data collection reports, see the topic *System Data Collection Set Reports* in the SQL Server Technical Documentation:

System Data Collection Set Reports

https://aka.ms/Euotdi

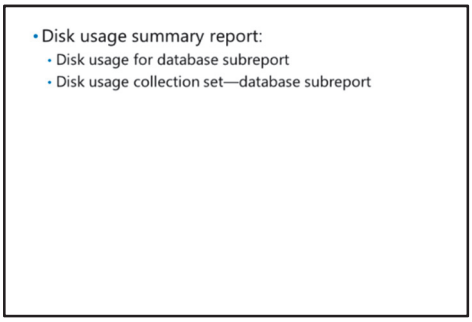
For information about how to access a data collection set report, see the topic *View a Collection Set Report (SQL Server Management Studio)* in the SQL Server Technical Documentation:

View a Collection Set Report (SQL Server Management Studio)

https://aka.ms/Wfn1u1

Disk Usage Reports

The Disk Usage reports are based on the Disk Usage system data collection set. By default, the collection set gathers disk usage data every six hours and keeps the data for two years.



Note that by default, the Disk Usage system data collection set retains data for much longer than other system data collection sets. However, because the amount of data that this collection set gathers is quite small, this should not be a concern. The value of being able to track individual file space usage over time warrants the use of this longer retention period.

This report also includes a number of hyperlinks that lead to a set of linked subreports that provide a more detailed view of the usage data:

- **Disk usage for database**. This subreport is displayed when you click a database name in the Disk Usage summary report. It shows pie charts of data file usage for a single database.
- **Disk usage collection set - database**. This subreport is displayed when you click a trend link in the Disk Usage summary report. It shows the historical data that the Disk Usage data collection set has gathered for a single database, together with a chart that maps the percentage of free space in the database data files.

Demonstration: Viewing the Disk Usage Report

In this demonstration, you will see how to:

- Force a data collection.
- Access the Disk Usage data collection report.

Demonstration Steps

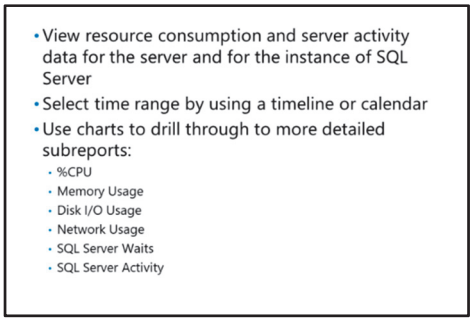
Force a Data Collection

1. In SQL Server Management Studio, in Object Explorer, under **Management**, under **Data Collection**, under **System Data Collection Sets**, right-click **Disk Usage**, and then click **Collect and Upload Now**.
2. In the **Collect and upload Data Collection Set** dialog box, click **Close**.
3. In Object Explorer, right-click **Query Statistics**, and then click **Collect and Upload Now**.
4. In the **Collect and upload Data Collection Set** dialog box, click **Close**.
5. In Object Explorer, right-click **Server Activity**, and then click **Collect and Upload Now**.
6. In the **Collect and upload Data Collection Set** dialog box, click **Close**.

Access the Disk Usage Report

1. In Object Explorer, under **MIA-SQL**, expand **Databases**, right-click **MDW**, point to **Reports**, point to **Management Data Warehouse**, and then click **Management Data Warehouse Overview**.
2. Review the hyperlinks under each report name that show the last data collection date and time. Under **Disk Usage**, click the date and time hyperlink.
3. In the **Disk Usage Collection Set** report, observe the data that is available in the report, and then click **InternetSales**.
4. In the **Disk usage for database: InternetSales** report, observe the available information, and in the report pane, in the upper-left corner, click the **Navigate Backward** button to return to the **Disk Usage Collection Set** report.
5. In the **Log Trend** column, click the trend line for the **MDW** database. Note that you must click the line itself; clicking elsewhere in the cell that contains the line will not work.
6. In the **Disk Usage Collection Set - Log: [MDW]** report, observe the information that is available in the report. Note that the chart in this report is based on percentage of free space in the data file.
7. Click the **Navigate Backward** button to return to the **Disk Usage Collection Set** report.
8. In the report pane, click the **Navigate Backward** button to return to the **Management Data Warehouse Overview: MDW** report.
9. Leave SQL Server Management Studio open for the next demonstration.

Server Activity Report



The Server Activity report is based on the Server Activity system data collection set. The collection set gathers statistics that are related to SQL Server, such as waits, locks, latches, and memory statistics that can be accessed by using DMOs. In addition, the collection set gathers Windows and SQL Server performance counters to retrieve information, such as CPU and memory usage from the system, and from the processes that are running on the system. The collection runs every 60 seconds and is uploaded every 15 minutes by default. By default, the history is retained for 14 days.

This report has a large number of linked subreports that provide much deeper information than appears on the initial summary. The initial report is a dashboard that provides an overview. If you investigate this report, you will find that almost every item that is displayed is a link to a more detailed subreport. For example, you can click a trend line in a graph to find out the values that make up the trend.

By default, the main report shows data for the whole data collection period. You can restrict your view to a time range or an individual data collection instance by using the timeline control at the top of the report or alternatively by using the **Calendar** button.

The report has six charts, each of which reports on a different aspect of server activity. You can click the data lines on four of the charts to drill through to a more detailed report:

- **%CPU**
- **Memory Usage**
- **Disk I/O Usage**
- **Network Usage**

For the remaining two charts, click the chart title to drill through to a more detailed view:

- **SQL Server Waits**
- **SQL Server Activity**

Demonstration: Viewing the Server Activity Report

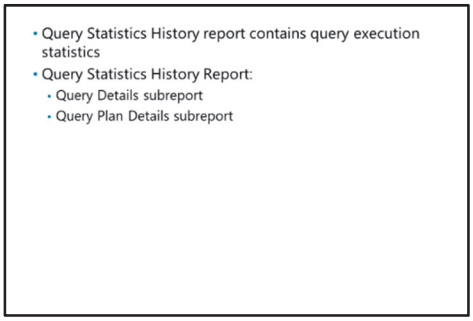
In this demonstration, you will see how to access the Server Activity data collection report.

Demonstration Steps

1. In SQL Server Management Studio, in the **Management Data Warehouse Overview** pane, in the report pane, in the upper-left corner, click the **Refresh** button.
2. In the main report, under **Server Activity**, click the date and time hyperlink.
3. In the **Server Activity History** report, observe the timeline and the six charts. Explain that some of the charts are empty because no relevant activity has taken place.
4. Demonstrate the use of the timeline; under the timeline, click the **Zoom In** button twice. Note how the range of dark blue squares, each representing a data collection point, reduces each time that you click, and the range of data in the charts changes to match the selected time range.

5. In the **Memory Usage** chart, click the lower line to drill through.
6. In the **SQL Server Memory Usage** report, scroll down to view the **SQL Server Internal Memory Consumption By Type** chart.
7. Expand the **Average Memory Use by Component** section of the report to view detailed memory usage information for each SQL Server component.
8. In the report pane, in the upper-left corner, click the **Navigate Backward** button to return to the **Server Activity History** report.
9. In the report pane, in the top left, click the **Navigate Backward** button five times to return to the **Management Data Warehouse Overview** report.
10. Leave SQL Server Management Studio open for the next demonstration.

Query Statistics Report



The Query Statistics report is based on the Query Statistics system data collection set that retrieves details of expensive queries. This collection set is the most intensive default collection system and, by default, runs every 10 seconds. To avoid the overhead of constant uploading, the data that this collection set gathers is cached in the local file system and uploaded by using SSIS every 15 minutes. The data that is collected is retained for 14 days by default—but this value can be extended.

SQL Server determines which queries are expensive based on:

- Duration
- CPU
- Logical writes
- Physical reads
- Execution count

You control the time range of the report by using a timeline in a similar fashion to the Server Activity report. For a given time range, you can view the top 10 most expensive queries by each of the cost metrics that are used to collect the data (such as duration and CPU).

This report also includes linked subreports that you can use to access higher levels of detail. As an example, you can retrieve query plans from the expensive queries that were in memory at the time that the capture was performed.

Query Details Subreport

This report provides a detailed view of the performance for an individual query over time, including a summary list of the query execution plan(s) that were used during the selected time period.

Query Plan Details Subreport

By drilling through from the Query Details subreport, you can view the details of individual query plans, including the graphical execution plan.

 **Note:** The data that the Query Statistics report returns has much in common with the Query Store—a new feature in SQL Server. The key difference is that Query Store is configured at database level, but the Query Statistics report covers one or more database engine instances in one report.

Demonstration: Viewing the Query Statistics Report

In this demonstration, you will see how to access the Query Statistics data collection report.

Demonstration Steps

1. In SQL Server Management Studio, in the **Management Data Warehouse Overview** pane, in the report pane, in the upper-left corner, click the **Refresh** button.
2. In the main report, under **Query Statistics**, click the date and time hyperlink.
3. In the **Query Statistics History** report, observe that the report uses the same timeline control as the **Server Activity** report to navigate through the data.
4. Under the **Navigate through the historical snapshots of data using the time line below**, click the  icon.
5. Under the **Top Queries by Total CPU** chart, click **Duration**, click **Total I/O**, click **Physical Reads**, and then click **Logical Writes** to demonstrate the different views of the data.
6. Click **CPU** to return to the original view.
7. Under the report, in the **Query** column of the data table, click the first row.
8. In the **Query Details** report, scroll down to view the various components of the report.
9. At the bottom of the report, in the **Top Query Plans By Average CPU Per Execution** table, in the **Plan #** column, click the first row.
10. In the **Query Plan Details** report, scroll down, and in the **Query Execution Statistics** section, click **View graphical query execution plan**.
11. Note that a new pane opens that contains the query plan.
12. Close SQL Server Management Studio without saving any changes.

Check Your Knowledge

Question	
Which system data collection set report would you use to get the history of memory usage?	
Select the correct answer.	
	The Server Activity report.
	The Query Statistics report.
	The Disk Usage report.

Lab: Monitoring SQL Server

Scenario

Part of your role in your organization is to ensure that all the companies' databases are running smoothly and efficiently. Your manager has had a complaint from the finance department that some of their reports have started to run slowly, especially during their month-end processes.

You have been tasked with ensuring that all the databases are configured to store performance data and are able to produce reports to provide proof that there are no health issues. Your manager is keen to be notified if during this process you uncover any issues that need resolving.

Objectives

After completing this lab, you will be able to:

- Create a Management Data Warehouse, and configure it to capture performance data.
- Produce performance reports.
- Analyze a Query Statistics Report to identify poorly performing queries.

Estimated Time: 30 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Create and configure a Management Data Warehouse

Scenario

The first step in configuring your SQL Server to capture and store performance data is to setup a data warehouse.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create a Management Data Warehouse

3. Capture data into the Management Data Warehouse

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20764C-MIA-DC** and **20764C-MIA-SQL** virtual machines are both running, and then log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab13\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, leave the window open as it is creating a load on the database.

▶ Task 2: Create a Management Data Warehouse

1. Use **Activity Monitor** to see a list of most expensive queries.
2. Note that the top most expensive query begins with **SELECT total.name**.
3. Use SQL Server Management Studio to create a new Management Data Warehouse named **MDW**.

▶ Task 3: Capture data into the Management Data Warehouse

1. Use SQL Server Management Studio to capture data into the **MDW** database.
2. Change the frequency of the data capture for Query Statistics to every 5 minutes every day.
3. View the Query Statics History report, and drill down to view the graphical query execution plan for the SELECT total.Name query.
4. Note there are no performance recommendations.

 **Note:** In a production environment you should consider the frequency of data you capture and the disk space required against the cost of storing that data.

Results: After completing this exercise, you should have configured a Management Data Warehouse called MDW on the MIA-SQL instance.

Exercise 2: Use the MDW to diagnose performance issues

Scenario

After creating the Management Data Warehouse you decided to investigate the performance of one of the top running queries run by the finance department.

You use a combination of the Activity Monitor and Management Data Warehouse to find a problem and make recommendations to your manager that will hopefully resolve their performance issues.

The main tasks for this exercise are as follows:

1. Prepare the lab environment

2. Investigate execution plans

▶ Task 1: Prepare the lab environment

1. In the **D:\Labfiles\Lab13\Starter** folder, right-click Execerise2.cmd, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**, leave the window open as it is creating a load on the database.

▶ Task 2: Investigate execution plans

1. Using Activity Monitor, view the execution plan for the most expensive query running against the database.
2. Has SQL Server identified any issues with the query? How would you resolve the issues?
3. Using the Management Data Warehouse, use the Query Statistics History report to view the Query Plan Details report.
4. Note that the report shows the same issues as the execution plan from Activity Monitor.

Results: At the end of this exercise you will be able to:

Use Activity Monitor to see which running queries are the most expensive

Question: What are the benefits of using a central data warehouse for SQL Server performance data, instead of local collection on each server?

Module Review and Takeaways

 **Best Practice:** In this module, you have seen different ways to monitor SQL Server Database Engine activity:

- Use DMOs to perform real-time monitoring and troubleshooting.
- Use Activity Monitor for easy access to the most relevant information.
- Use Performance Monitor to gather metrics for Windows and SQL Server.
- Create a central management data warehouse to hold historical performance information.

Review Question(s)

Question: Which SQL Server activity monitoring tools are best suited to your organization's needs?

Module 14

Troubleshooting SQL Server

Contents:

Module Overview

Database administrators working with Microsoft® SQL Server® need to adopt the important role of troubleshooter when issues arise—particularly if users of business-critical applications that rely on SQL Server databases are being prevented from working. It is important to have a solid methodology for resolving issues in general, and to be familiar with the most common issues that can arise when working with SQL Server systems.

Objectives

After completing this module, you will be able to:

- Describe a troubleshooting methodology for SQL Server.
- Resolve service-related issues.
- Resolve login and connectivity issues.

Lesson 1
A Troubleshooting Methodology for SQL Server

Before starting to try to resolve any issue, it's important to be prepared to apply a logical troubleshooting methodology in a consistent manner. Although troubleshooting is often regarded as an art as much as a science, there are a number of characteristics common to many good troubleshooters. You should aim to develop or emulate those characteristics to ensure that you can successfully troubleshoot issues in your organization.

Lesson Objectives

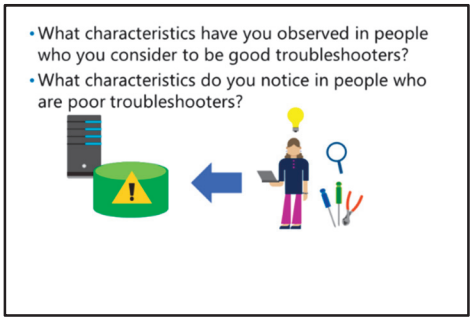
After completing this lesson, you will be able to:

- Describe the characteristics of a good troubleshooter.
- Apply a troubleshooting methodology.

Discussion: Characteristics of Good Troubleshooters

Consider the following questions:

- What characteristics have you observed in people who you consider to be good troubleshooters?
- What characteristics do you notice in people who are poor troubleshooters?



Characteristics of a good troubleshooter might include:

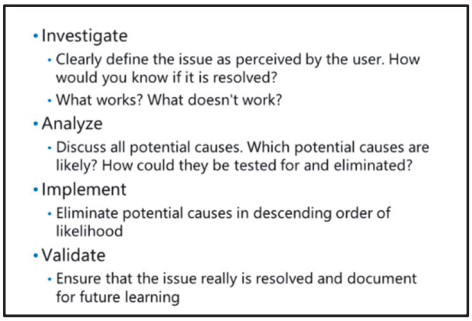
- Follows a repeatable and logical methodology.
- Stays calm, even in stressful situations.
- Is able to continually subdivide problems to move towards a solution.
- Knows about available tools and how to use them.
- Ensures that the problem really is resolved.
- Accepts responsibility for resolving a problem.

Characteristics of a poor troubleshooter might include:

- Adopts a haphazard approach that does not follow a logical path.
- Is inclined to panic under stress.
- Makes assumptions that are not supportable.
- Quickly decides upon a cause for problems—without justification or evidence—and then tries to justify the selection of the cause.
- Assumes problems are resolved without verification.
- Does not accept responsibility for resolving a problem.

Applying a Troubleshooting Methodology

A key characteristic of good troubleshooters is that they follow a clear methodology in a logical manner. There are many different methodologies that you can use, but the following list describes phases that are common to most of them:



Investigation Phase

This is a critical phase. Before you can solve any problem, you need to be very clear in your understanding of what the problem is. It can be very tempting to bypass this step and jump directly to attempting solutions, especially when a serious problem occurs or you are under pressure. However, acting without sufficient information is often a waste of effort, and might actually make the issue worse.

You need to understand what does and doesn't work before attempting to resolve a problem. Gather as much information as you can about the circumstances in which the issue occurred. Some questions you might ask include:

- **When did the issue occur?** Knowing the time and date that an issue occurred can help you find information from application and operating system log files, or might suggest that the problem has a link to another event.
- **In what environment did the error occur?** This is a broad category, which will vary a lot depending on the nature of the problem and the system you are troubleshooting. Environmental characteristics that might be helpful when troubleshooting include:
  - Application version number.
  - Operating system name and version number.
  - Internet browser name and version number.
  - The user's security context (application user name or database login).
- **What was the user doing when the issue occurred?** Knowing how the problem presents itself to the user can help you understand which part of a system, or which sections of code, might be causing the problem. One very important concept in this phase is that the issue needs to be defined from the affected user's point of view, not from the assumed perspective of an IT professional. For example, there is no point telling a user that a system is working if they cannot use it for any reason—regardless of how your IT-based perspective might tell you that the system is working.
- **Were any error messages displayed? If so, what was the error number and/or error text?** Application error messages can often give a clear indication of the cause of an issue. System errors from the SQL Server Database Engine often include a clear description of their causes.
- **Can the problem be reproduced, or was it a one-off? If the problem can be reproduced, what are the steps to take?** Troubleshooting is always easier if you know how to reproduce the issue, because this makes it much easier to investigate causes and test your fix.
- **How many users have experienced the problem?** If an issue affects a subset of users, the differences between affected and unaffected groups might suggest the cause of the problem.

- **When did the affected component last work correctly?** Sometimes, when a user complains that something doesn't work, it might be that it has never worked. Verify that there was a time when the system worked as expected, and find out when that was.
- **What changes have been made to the affected system since the component last worked correctly?** Troubleshooting is often the process of identifying the change that causes a working system to go into a broken state; knowing details of the changes that have occurred can help you identify the cause.

 **Note:** These questions are mainly phrased in terms of an end user reporting a problem. However, most of these questions are valid even when there is no end user of a system—for example, in a problem involving an automated process, such as a SQL Server Agent job.

Finally, you need to know how the user would decide that the issue is resolved. A common mistake when troubleshooting is to find a problem, assume that it is the cause of the issue, resolve that problem, and then conclude that the original issue is now settled—without verifying that the original issue is actually resolved.

Analysis Phase

In the analysis phase, you need to determine all the possible causes of the issue that you are trying to resolve. At this point, it is important to avoid excluding any potential causes, no matter how unlikely you consider them to be. The information you gather in the investigation phase might suggest potential causes.

A discussion with one or more other people is often useful in this phase, particularly if you can obtain alternative viewpoints. The analysis phase often benefits from two types of people—one with an excellent technical knowledge of the product and another who constantly requires the first person to justify their thoughts, while thinking both logically and laterally.

Implementation Phase

In the implementation phase, you need to eliminate each potential cause. This process of elimination usually returns the best results when the potential causes are ruled out in order, from the most likely to the least likely. The information you gathered in the investigation phase will help you assess the relative likelihood of possible causes; a better possible cause explains the behavior you have observed, and the more likely it is to be the cause of a problem.

The critical aspect of the implementation phase is to make sure that your reasons for eliminating potential causes are logically valid. If you reach the end of your list of potential causes and have not yet found a solution to the issue, you should return to the analysis phase and recheck your thinking. If you cannot find a problem in your analysis, you might need to recheck your initial assumptions in the investigation phase, or gather more information.

Validation Phase

It is easy, particularly when you are new to troubleshooting, to assume that problems are resolved when they are not. Do not assume that, because you have found and resolved a problem, this was the original one that you were aiming to solve.

In the investigation phase, you should have determined how the user would decide if the issue is resolved. In the validation phase, you must apply that test to see if the issue really is resolved.

Documentation

After the problem is resolved, it is good practice to document your findings. This means that other members of your organization can understand and learn from the issue, and also provides a useful guide to resolving the issue should it occur again in the future.

Sequencing Activity

Number each of the following troubleshooting phases to indicate their correct order.

	Steps
	Investigation Phase
	Analysis Phase
	Implementation Phase
	Validation Phase
	Create Documentation

Lesson 2
Resolving Service-Related Issues

In the remainder of this module, you will see how to approach common types of issues that can arise when working with SQL Server systems.

SQL Server comprises several Windows® services. While troubleshooting these services has much in common with troubleshooting all Windows services, there are some considerations specific to SQL Server. This lesson covers the types of issue involving problems with SQL Server services.

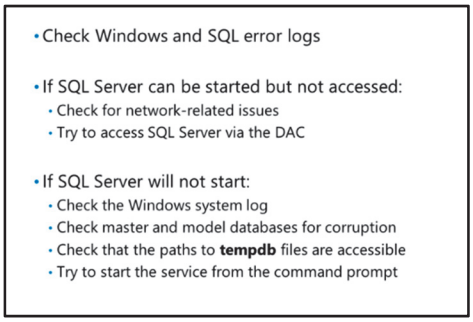
Lesson Objectives

After completing this lesson, you will be able to:

- Troubleshoot service-related issues.
- Use the SQL Server error log.
- Use Windows event logs.

Troubleshooting Service-Related Issues

The most common service-related problem is when one of the SQL Server services will not start or cannot be accessed. As with all Windows services, the SQL Server services are not permitted to interact directly with the system console. This limits the options that are available to the service to advise you of problems. Also, it is uncommon for a user to be logged on at the console to receive any notifications, even if they have sufficient permission to do so.



Windows and SQL Server Logs

The most common cause of a service not being able to start is that the service account logon is failing for some reason. This may be due to an incorrect or expired password, or the account could be locked out. Logon failures for services typically appear in the Windows System Event Log. More complex issues, such as missing files, appear in the SQL Server logs. It is recommended, therefore, that you check both the Windows and SQL Server logs as the first step in resolving service startup issues.

Other Service-Related Issues

If SQL Server does start but cannot be accessed, the problem is likely to relate to network issues. Check that the necessary network protocols are enabled for the SQL Server service, and that firewall rules permit connections to the relevant ports.

In rare cases, the SQL Server scheduler can hang and stop accepting new connections. In this and similar situations, you should also try to connect to SQL Server by using the Dedicated Administration Connection (DAC) to access the instance and analyze the problem. Occasionally, you might need to kill individual sessions that are causing the problem.

For more information about using the DAC, see the topic *Diagnostic Connection for Database Administrators* in the SQL Server Technical Documentation:

Diagnostic Connection for Database Administrators

https://aka.ms/Ihvyn9

If SQL Server will not start but the issue is not caused by a problem with the service account, check:

- Whether the SQL Server log files indicate that the **master** or **model** database is corrupt. If either is corrupt, follow the procedures to recover the databases.
- Whether the file paths to the **tempdb** database files are accessible. SQL Server recreates the **tempdb** database each time the server instance starts, but the path to the database files (as configured in the master database) must exist and be accessible to the service account under which the SQL Server service is running.
- Whether you can start the instance by using the command prompt. If starting SQL Server from a command prompt does work, check the configuration of the service and make sure that the permission requirements are met.

For more information about the steps to rebuild system databases, see the topic *Rebuild System Databases* in the SQL Server Technical Documentation:

Rebuild System Databases

https://aka.ms/Ajfk4g

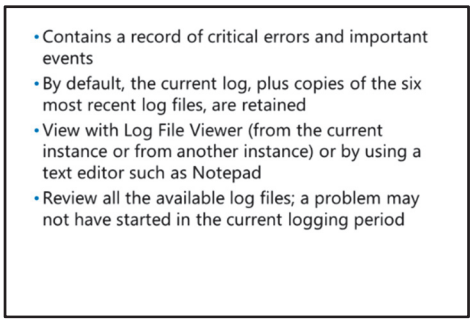
For more information about starting the SQL Server Database Engine from the command prompt, see the topic *sqlservr Application* in the SQL Server Technical Documentation:

sqlservr Application

https://aka.ms/Schfvr

SQL Server Error Log

The SQL Server log will often provide detailed information about startup issues—in addition to information about many other issues. You can view the log by using the Log File Viewer, which is part of SQL Server Management Studio (SSMS). However, if you cannot start the SQL Server service, you are unable to use the log viewer to see the contents of the SQL Server logs directly. You can, however, open these logs from another SQL Server instance, or by using a text editor, such as Notepad.



By default, SQL Server writes its error logs to the ERRORLOG file in the LOG folder in the servers specific instance within the Program Files folder. The log file location is passed to the SQL Server service as a startup parameter; you can customize it by changing the service startup parameter, either when SQL Server is installed, or by changing the parameter through SQL Server Configuration Manager.

SQL Server keeps a set of archive log files that you can also review, because problems may have been occurring for some time. By default, backups of the previous six log files are retained. The most recent log archive has the extension .1, the second most recent the extension .2, and so on. The current error log has no extension. You can increase the number of log files to retain by customizing the log configuration, but you cannot choose to retain fewer than six log files.

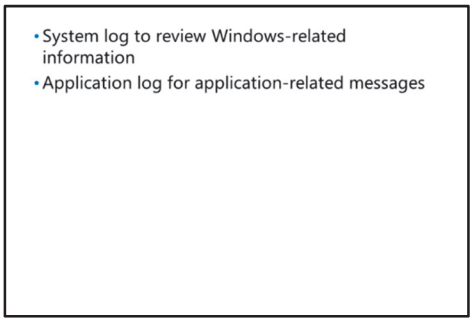
For more information about viewing SQL Server log files from a remote instance, see the topic *View Offline Log Files* in the SQL Server Technical Documentation:

View Offline Log Files

https://aka.ms/S7m853

Windows Event Logs

Because Windows services cannot interact directly with the console, the key place to find information about issues is the Windows System Event Log. As with many other applications, SQL Server writes a substantial amount of information to the Windows application log. Therefore, you should check both these logs periodically, in case errors are being logged but not producing symptoms apparent to users.



Note: In some organizations, the security policy does not allow database administrators to have access to Windows event logs. If this is the case in your organization, you will need the assistance of Windows administrators to examine Windows logs.

For instructions about how to access Windows logs, see the topic *View the Windows Application Log (Windows)* in the SQL Server Technical Documentation:

View the Windows Application Log (Windows 10)

https://aka.ms/Lu1s3x

To reduce the number of log events shown in the log viewer, apply a filter to include only those events with an event level of **Critical**, **Error,** or **Warning**. This will restrict your view to only the most significant events.

Note: When you are reviewing Windows event log data, be conscious that errors or warnings not generated by SQL Server services can have an impact on SQL Server's behavior. For example, a message from a storage controller indicating that a drive in a RAID array has failed might indicate a problem with SQL Server's I/O performance; or a message about an incorrectly configured network adapter might indicate that query results will be delivered more slowly to client applications.

Demonstration: Troubleshooting Service-Related Issues

In this demonstration, you will see how to troubleshoot a service-related issue.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **20764C-MIA-DC**, and **20764C-MIA-SQL** virtual machines are running, and log on to **20764C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod14** folder, run **Setup.cmd** as Administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. Click the **Start** button, then type **Configuration Manager**, click **SQL Server Configuration Manager**.
5. In the **User Account Control** dialog box, click **Yes**.
6. In SQL Server Configuration Manager, in the left pane, click **SQL Server Services**.
7. In right pane, note that the **SQL Server (SQL2)** service is not running.
8. Right-click **SQL Server (SQL2)** and click **Start**. Note that the service does not start successfully, and an error message is returned.
9. In the **SQL Server Configuration Manager** dialog, click **OK**.
10. To check the Windows system log, click **Start**, type **Event Viewer**, and press Enter.
11. In Event Viewer, in the left pane, expand **Windows Logs**, and then click **System**.
12. Click on the most recent message with a **Level** of **Error** and a **Source** of **Service Control Manager**.
13. On the **Details** tab, note that the error message states that there is a service specific error, and provides only the following details:

```
SQL Server (SQL2)
%%17113
```

14. Close the Event Viewer window.
15. To check the SQL Server error log, start File Explorer, and navigate to **C:\Program Files\Microsoft SQL Server\MSSQL14.SQL2\MSSQL\Log**.
16. If a **Log** dialog box appears, click **Continue**.
17. Right-click **ERRORLOG**, click **Open with**, and then click **Notepad**. Notice the last three lines of the file include the error number displayed in the Windows system log (17113), and a detailed description of the problem (scroll to the right to read the full error message). The message indicates that the data file for the **master** database cannot be found.
18. In File Explorer, navigate to the folder location mentioned in the error message (**C:\Program Files\Microsoft SQL Server\MSSQL14.SQL2\MSSQL\DATA**).
19. If a **DATA** dialog box appears, click **Continue**. Notice that the folder contains the file **master.AV0001**. The demonstration simulates the situation where the **master.mdf** file has been quarantined by an antivirus application, which has renamed it **master.AV0001**.

    (In a real-world scenario, you would need to recover the file from the quarantine system, and prevent the antivirus application from scanning this folder, before attempting to restart the service.)

20. For the purposes of this demonstration, rename the file **master.AV0001** to **master.mdf**; right-click **master.AV0001** then click **Rename**.
21. Replace the **AV0001** file extension with **mdf**, then press Enter.
22. In the **Rename** dialog box, click **Yes**.
23. Close File Explorer.
24. In SQL Server Configuration Manager, in the right pane, right-click **SQL Server (SQL2)**, and then click **Start**. Note that the service starts successfully.
25. Close SQL Server Configuration Manager and Notepad.

Check Your Knowledge

Question	
Which log(s) will give you the most information when the SQL Server service will not start?	
Select the correct answer.	
	The Windows system log.
	The Windows application log.
	The SQL Server error log.
	The SQL Server error log and the Windows system log.
	The SQL Server error log and the Windows application log.

Lesson 3
Resolving Connectivity and Login Issues

Database administrators who work with SQL Server are commonly called upon to resolve problems with connectivity and logins. Users might have problems making connections to SQL Server and, in addition to resolving problems with network names, protocols, and ports, you might also have to investigate issues with logins and passwords.

Best Practice: Using logins based on Windows authentication removes the need for you, as a database administrator, to deal with most password and authentication-related issues.

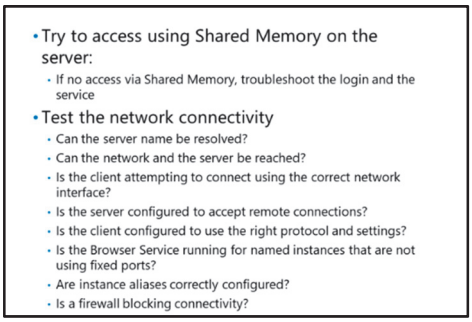
Lesson Objectives

After completing this lesson, you will be able to:

- Troubleshoot connectivity issues.
- Troubleshoot login failures.

Troubleshooting Connectivity Issues

A connectivity problem will typically involve a situation where one or more users or client applications is unable to connect to a database engine instance.



Eliminate a Network Problem

The first step in trying to resolve a connectivity issue should be to determine whether it is related to network connectivity. If clients cannot connect to a database engine instance because the SQL Server service cannot start, you must resolve any problems with the service before looking for network connectivity problems. You can quickly identify whether a network problem might be causing connectivity issues by attempting to connect to the instance via a shared memory connection from the server console. If you cannot access SQL Server via a shared memory connection, you will need to troubleshoot the login (as discussed later in this lesson) or the service (as discussed in the previous lesson). Error messages received by the client typically indicate whether the initial connection to the database engine instance is successful.

Network-Related Issues

If a login attempt using shared memory succeeds, the problem is likely to be network-related. In rare cases, you may see a problem with incompatible network libraries on the client and the server, but most problems are much less subtle. The following table lists some tests for common causes of network-related connectivity issues:

Test	Action
Can the server name be resolved?	If the client application is attempting to connect to a database engine instance by name, it must be possible for the client to resolve the server's network name to a routable address. In TCP/IP networks, this will typically require a DNS lookup. You can use command-line tools such as **ping** and **nslookup** to confirm whether the server name can be correctly resolved to an IP address, and to find out the DNS server being used to resolve DNS queries. Another indicator of name resolution problems is that you can connect to an instance by specifying the server IP address, but an attempt to connect using the server name fails.
Can the network and the server be reached?	While a server name might be successfully resolved to a network address, no connectivity may be possible between the client and server systems. On TCP/IP networks, you might be able to test the network route from the client to the server using command-line tools such as **ping** or **tracert**. However, you should be aware that, on some networks, the ICMP protocol used by these tools is disabled, in which case the tests will be inconclusive. Problems in this area might indicate that changes need to be made to the configuration network devices, such as switches and gateways.
Is the client attempting to connect on the correct network interface?	On a server with more than one network interface, SQL Server might not be configured to accept connections on all of them. You can verify which network interfaces a database engine instance is configured to use through SQL Server Configuration Manager. If an instance is not listening on all of a server's network interfaces, verify that the client is attempting to connect to an interface being used by SQL Server.
Is the server configured to accept remote connections?	Remember that, by default, a new SQL Server instance has no network protocols enabled. You will not be able to make connections to a database engine instance over Named Pipes or TCP/IP until you enable the protocol using SQL Server Configuration Manager.
Is the client configured to use the right protocol and settings?	Check that the network protocol that the client is attempting to use to connect to the database engine instance is enabled on the server. The client protocol will typically be defined in the client application connection string. Server settings can be viewed using SQL Server Configuration Manager.

Test	Action
Is the SQL Server Browser service running for named instances that are not using fixed ports?	If you are using named instances of SQL Server, the client systems need to be able to resolve the names of the instances to a port number. By default, this is achieved by connecting to the SQL Server Browser service on UDP port 1434. Check whether the SQL Server Browser service is running, and whether database engine instances are configured to use dynamic TCP/IP ports, through SQL Server Configuration Manager. Be aware that, even if the SQL Server Browser service is running, individual database engine instances can be configured not to be advertised by the SQL Server Browser service using the **HideInstance** flag in the server network configuration. If you do not want to run the SQL Server Browser service, assign fixed ports to named instances, and consider configuring client aliases.
Are instance aliases correctly configured?	If an alias is configured for a database engine instance, confirm that it exists for the network library used by the client application. Remember that aliases are defined separately for 32-bit and 64-bit clients. Check alias configuration using SQL Server Configuration Manager.
Is a firewall blocking connections?	Check to make sure that there is no firewall blocking the ports that you are trying to connect over; this might be a network firewall, or the Windows firewall on the client or server. If a firewall is blocking your access, an exception or rule will likely need to be configured in the firewall.

 **Best Practice:** When you suspect a network connectivity problem in a TCP/IP network, start by testing a connection from the client to the server using the server IP address—and, if necessary, the TCP port number. Should a connection by IP address fail, the issue is likely to be with the network—perhaps a routing or firewall problem. If a connection by IP address is successful but a connection by name fails, the issue is likely to be with name resolution—either DNS, the SQL Server Browser service, or SQL Server aliases.

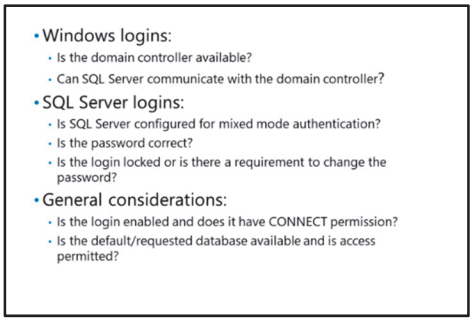
For more information about configuring SQL Server networking, see the topic *Server Network Configuration* in the SQL Server Technical Documentation:

 **Server Network Configuration**

https://aka.ms/qogu0b

Troubleshooting Login Failures

A login problem typically displays symptoms that show a client application that can establish a network connection to SQL Server—but cannot complete a successful logon. The troubleshooting actions that you need to perform depend upon the type of login being used:



- For Windows logins:
  - Make sure that SQL Server can connect to a domain controller to authenticate Windows accounts.
  - Inspect the Windows logs to review potential issues.
- For SQL Server logins:
  - Make sure that the database engine instance is configured for SQL Server authentication. When SQL Server is configured for Windows authentication only, SQL Server logins can still be created and enabled, but cannot be used to connect to the instance. The most common error is returned when a SQL Server login is used but, if SQL Server authentication is disabled, this indicates that a trusted connection is not available.
  - Make sure that the supplied password is correct.
  - Make sure that the login has not been locked out by an account policy. If a SQL Server login is configured to use a password expiry policy—but the client application is not configured to handle the exchange of messages required to change an expired policy—the application will stop working when the login's password expires. This situation is common with applications that were designed for versions of SQL Server that do not implement account policy for SQL Server logins.

For both Windows and SQL Server logins, make sure that the login has permission to connect to SQL Server, in addition to being able to access the database it is attempting to connect to. This check should include the default database for the login.

If a login problem is happening with a large number of different users, check for a failing logon trigger. When a logon trigger prevents users from connecting, the error message that is returned to interactive users indicates that a logon trigger prevented the connection.

When a login issue occurs, the error message returned to the client application is often generic, indicating that a login issue occurred, but not showing the details of the problem. A good example of this is SQL Server error number 18456, which is raised for a number of different login issues, and returns the following error message to the client application:

```
Login failed for user '<user_name>'. (Microsoft SQL Server, Error: 18456)
```

This is a security measure, designed to limit the leakage of information about server logins to an attacker, but this can make troubleshooting more difficult. In most instances of a login problem, a message is written to the SQL Server error log containing more information about the problem. In the case of error number 18456, the **State** property of the error message written to the SQL Server error log indicates the cause of the error. A detailed error message is also written to the log.

For more information about the **State** values returned with error number 18456, see the topic *MSSQLSERVER_18456* in the SQL Server Technical Documentation:

MSSQLSERVER_18456

http://aka.ms/n2n2j6

Check Your Knowledge

Question	
You want to define an alias for a named instance of the database engine. The alias will be used by clients using both 32-bit and 64-bit native client drivers. In SQL Server Configuration Manager, where should you define the alias?	
Select the correct answer.	
	Under SQL Native Client 11.0 Configuration.
	Under SQL Native Client 11.0 Configuration (32-bit).
	Under both SQL Native Client 11.0 Configuration (32-bit) and SQL Native Client 11.0 Configuration.

Lab: Troubleshooting Common Issues

Scenario

You need to be able to resolve common issues with SQL Server processes and services at the time they are occurring. This lab contains five exercises that cover different types of issue—you should attempt to troubleshoot and resolve as many of them as possible.

Objectives

At the end of the lab, you will be able to troubleshoot and resolve:

- SQL login issues
- Service issues
- Windows login issues
- Job execution issues
- Performance issues

Estimated Time: 75 minutes

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

User name: **AdventureWorks\Student**

Password: **Pa55w.rd**

Exercise 1: Troubleshoot and Resolve a SQL Login Issue

Scenario

Users of the Promote application are complaining that it can no longer connect to the server. The application connects using the SQL login **PromoteApp**.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Review the Exercise Scenario

3. Troubleshoot and Resolve the Issue

▶ Task 1: Prepare the Lab Environment

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:\Labfiles\Lab14\Starter** folder as Administrator.

▶ Task 2: Review the Exercise Scenario

- Review the exercise scenario; make sure that you understand the problem you are trying to resolve.

▶ Task 3: Troubleshoot and Resolve the Issue

- Use SSMS to investigate and resolve the problem with the **PromoteApp** login.

Results: After this exercise, you will have investigated and resolved a SQL login issue; the **PromoteApp** login will be functioning properly.

Exercise 2: Troubleshoot and Resolve a Service Issue

Scenario

Users are reporting that the Reporting Services instance at http://mia-sql/Reports_SQL2/Pages/Folder.aspx is not accessible; an error message with the following text is returned:

```
The report server cannot open a connection to the report server database. A
connection to the database is required for all requests and processing.
(rsReportServerDatabaseUnavailable)
For more information about this error navigate to the report server on the local
server machine, or enable remote errors
```

This instance of Reporting Services uses databases hosted by the **MIA-SQL\SQL2** database engine instance.

The main tasks for this exercise are as follows:

1. Review the Exercise Scenario

2. Troubleshoot and Resolve the Issue

▶ Task 1: Review the Exercise Scenario

- Review the exercise scenario; make sure that you understand the problem you are trying to resolve.

▶ Task 2: Troubleshoot and Resolve the Issue

Gather More Information

1. Using Internet Explorer, browse to **http://mia-sql/Reports_SQL2/Pages/Folder.aspx**. This will give you a more detailed error message than users have reported.
2. Use Event Viewer to check the Windows system log for messages relating to the **MIA-SQL\SQL2** service.

Resolve the Issue

1. The network administrator has notified you that the password for the **ADVENTUREWORKS\ServiceAcct** was recently changed to **Pa55w.rd**. This information might help you to resolve the problem.
2. Test your fix using Internet Explorer.

Results: After this exercise, the Reporting Services instance at http://mia-sql/Reports_SQL2/Pages/Folder.aspx will be functioning properly.

Exercise 3: Troubleshoot and Resolve a Windows Login Issue

Scenario

A Windows user (ADVENTUREWORKS\AnthonyFrizzell) is trying to connect to the **MIA-SQL** instance to execute some Transact-SQL queries using the **sqlcmd** application. The user reports that, whenever he attempts to connect, he receives the following error message:

```
Sqlcmd: Error: Microsoft ODBC Driver 11 for SQL Server : Login failed for user
'Adventureworks\AnthonyFrizzell'
```

The main tasks for this exercise are as follows:

1. Simulate the Issue
2. Review the Exercise Scenario
3. Troubleshoot and Resolve the Issue

▶ Task 1: Simulate the Issue

- Run **FrizzellQuery.cmd** in the **D:\Labfiles\Lab14\Starter** folder. Once the script completes, press any key to close the command prompt window.

▶ Task 2: Review the Exercise Scenario

- Review the exercise scenario; make sure you understand the problem you are trying to resolve.

▶ Task 3: Troubleshoot and Resolve the Issue

1. Verify that a login exists for **ADVENTUREWORKS\AnthonyFrizzell** on the **MIA-SQL** instance, and that the login is correctly configured and enabled.
2. Use the SQL Server error log to try and determine the cause of the user's connectivity problem.
3. What should the user do to resolve the connectivity problem?

Results: At the end of this exercise, you will be able to explain to the user why he cannot connect to the database.

Exercise 4: Troubleshoot and Resolve a Job Execution Issue

Scenario

Users are complaining that the **Get File List** SQL Server Agent job is failing to execute.

The main tasks for this exercise are as follows:

1. Review the Exercise Scenario
2. Execute the Failing Job
3. Troubleshoot and Resolve the Issue

▶ Task 1: Review the Exercise Scenario

- Review the exercise scenario; make sure that you understand the problem you are trying to resolve.

▶ Task 2: Execute the Failing Job

- Using SSMS, execute the **Generate File List** job from the **MIA-SQL** instance of SQL Server Agent.

▶ Task 3: Troubleshoot and Resolve the Issue

- Review the job history for the **Generate File List** SQL Server Agent job. Use the information you find in the log to troubleshoot and resolve the issue.

Results: After this exercise, you will have investigated and resolved a job execution issue.

Exercise 5: Troubleshoot and Resolve a Performance Issue

Scenario

Users of the **SalesSupport** application are reporting that the application has suddenly stopped responding; the **SalesSupport** administrator asks you to investigate, because the application log contains several recent messages reporting that queries run against the **InternetSales** database have timed out.

The main tasks for this exercise are as follows:

1. Simulate the Issue
2. Review the Exercise Scenario
3. Troubleshoot and Resolve the Issue

▶ Task 1: Simulate the Issue

- Run **Performance.cmd** in the **D:\Labfiles\Lab14\Starter** folder. The script will start three other command prompt windows that represent users of the **InternetSales** database. Do not interact with or close these windows; two of them will close when the issue is resolved.

▶ Task 2: Review the Exercise Scenario

- Review the exercise scenario; make sure that you understand the problem you are trying to resolve.

▶ Task 3: Troubleshoot and Resolve the Issue

1. Verify that the **InternetSales** database is working and accessible for new connections.
2. Determine the cause of the issue and resolve it. When the issue is resolved, two of the command prompt windows that were started in the first task of this exercise will close.

   If you are uncertain how to proceed, check Activity Monitor for the **MIA-SQL** database engine instance.
3. Once the issue is resolved, close all the application windows opened on **20764C-MIA-SQL**.

Results: At the end of this exercise, you will have resolved a performance issue.

Question: What tools might you use to monitor an intermittent or long-term issue?

Module Review and Takeaways

In this module, you have learned about methods and tools you might use when investigating issues with the SQL Server Database Engine; you will be able to build upon this to develop your troubleshooting skills. You have discussed the characteristics of successful troubleshooters, and have seen an example of a troubleshooting methodology. You have also seen and worked with several different sources of information, including log files and SQL Server tools, which you can use to understand and trace the causes of issues.

Review Question(s)

Question: How do you rate your troubleshooting skills? What could you do to improve them?

Tools

Participating in online forums, where developers and administrators post questions about SQL Server issues, is a good way to practice your troubleshooting skills, and to gain insight into methods used by other troubleshooters.

Module 15

Importing and Exporting Data

Contents:

Module Overview

While a great deal of data residing in a Microsoft® SQL Server® system is entered directly by users who are running application programs, there is often a need to move data in other locations, to and from SQL Server.

SQL Server provides a set of tools you can use to transfer data in and out. Some of these tools, such as the **bcp** (Bulk Copy Program) utility and SQL Server Integration Services, are external to the database engine. Other tools, such as the BULK INSERT statement and the OPENROWSET function, are implemented in the database engine. With SQL Server, you can also create data-tier applications that package all the tables, views, and instance objects associated with a user database into a single unit of deployment.

In this module, you will explore these tools and techniques so that you can import and export data to and from SQL Server.

Objectives

After completing this lesson, you will be able to:

- Describe tools and techniques for transferring data.
- Import and export table data.
- Use **bcp** and BULK INSERT to import data.
- Use data-tier applications to import and export database applications.

Lesson 1
Transferring Data to and from SQL Server

The first step in learning to transfer data in and out of SQL Server is to become familiar with the processes involved, and with the tools that SQL Server provides to implement data transfer.

When large amounts of data have to be inserted into SQL Server tables, the default settings for constraints, triggers, and indexes are not likely to provide the best performance possible. You might achieve improved performance by controlling when the checks that are made by constraints are carried out and when the index pages for a table are updated.

Another approach is to load data into a staging or work table, process it, then move the processed data into its final location. Partition switching offers one method of achieving this.

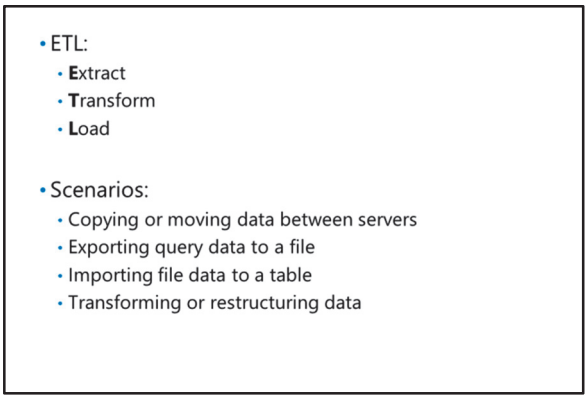
Lesson Objectives

After completing this lesson, you will be able to:

- Describe core data transfer concepts.
- Describe the tools that SQL Server provides for data transfer.
- Improve the performance of data transfers.
- Disable and rebuild indexes.
- Disable and enable constraints.
- Use partition switching as part of a data transfer.

Overview of Data Transfer

Not all data can be entered row-by-row by database users. Often data has to be imported from external data sources, such as other database servers or files. Also, users often request that data from tables in databases is exported to text files.



Data Transfer Steps

Although not all data transfer requirements are identical, there is a standard process that most data transfer tasks follow. The main steps are:

- Extracting data from a given data source.
- Transforming the data in some way to make it suitable for the target system.
- Loading the data into the target system.

Together, these three steps are commonly referred to as an Extract, Transform, Load (ETL) process, which can be implemented by the use of ETL tools.

Extracting Data

While there are other options, extracting data typically involves executing queries on a source system to retrieve the data, or opening and reading source files.

During the extraction process, there are two common aims:

- To avoid excessive impact on the source system. For example, do not read entire tables of data when you only have to read selected rows or columns. Also, do not continually re-read the same data, and avoid the execution of statements that block users of the source system in any way.
- To ensure the consistency of the data extraction. For example, do not include one row from the source system more than once in the output of the extraction.

Transforming Data

The transformation phase of an ETL process will generally involve several steps, such as the following:

- Data might have to be cleansed. For example, you might want to remove erroneous data, eliminate duplicates, or provide default values for missing columns.
- Lookups might have to be performed. For example, the input data might include the name of a customer, but the database might need an ID for the customer.
- Data might have to be aggregated. For example, the input data might include every transaction that occurred on a given day, but the database might require only daily summary values.
- Data might have to be de-aggregated. This is often referred to as data allocation. For example, the input data might include quarterly budgets, but the database might require daily budgets.

In addition to these common operations, data might have to be restructured in some way—for example, by pivoting the data so that columns become rows, concatenating multiple source columns into a single column, or splitting a single source column into multiple columns.

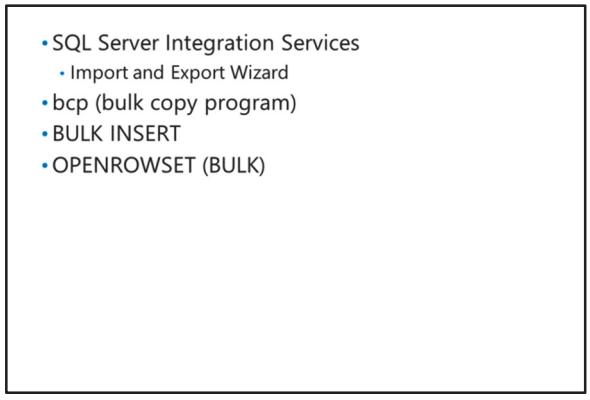
 **Note:** Data transformation is often most complex when you do not have control over the data extraction phase—for example, when data files are provided to you in a fixed format by a third party.

Loading Data

After data is transformed into an appropriate format, you can load it into the target system. Instead of performing row-by-row insert operations for the data, you can use special options for loading data in bulk. Additionally, you can make temporary configuration changes to improve the performance of the load operation.

Data Transfer Tools

SQL Server provides a set of tools for performing data transfer tasks. It is important to understand which tool is best to use for which types of scenario.



Note: You are likely to encounter requirements—for example, importing data from a text file into a SQL Server table—which could be accomplished with any of the tools listed here.

SQL Server Integration Services

SQL Server Integration Services (SSIS) is an ETL tool that ships with SQL Server. SSIS is capable of connecting to a wide variety of data sources and destinations, and can perform complex transformations on data. SSIS includes many tasks and transformations, and can also be extended by the use of custom .NET Framework components and scripts.

SQL Server Import and Export Wizard

SQL Server provides the Import and Export Wizard, which is a simple method of creating SSIS packages.

Bulk Copy Program (bcp)

You can use the Bulk Copy Program (**bcp**) to import large numbers of new rows from an operating system data file into a SQL Server table, or to export data from a SQL Server table to an operating system file. Although you can use the **bcp** utility with the **queryout** option—with which you can specify the rows to be exported with a Transact-SQL query—the normal use of **bcp** does not require any knowledge of Transact-SQL.

BULK INSERT

You can use the BULK INSERT Transact-SQL statement to import data directly from an operating system data file into a database table. Although the configuration options for BULK INSERT and **bcp** are similar, BULK INSERT differs from **bcp** in a number of ways:

- You execute the BULK INSERT statement from within Transact-SQL, whereas the **bcp** utility is a command-line utility.
- While the **bcp** utility can be used for both import and output, the BULK INSERT statement can only be used for data import.

OPENROWSET (BULK)

OPENROWSET is a table-valued function that you can use to connect to and retrieve data from OLE-DB data sources. Full details of how to connect to the data source must be provided as parameters to the OPENROWSET function. You can use OPENROWSET to connect to other database engines and data providers for which a driver is installed on the Windows® installation where SQL Server is installed.

A special OLE-DB provider called BULK is available for reading data from text files with the OPENROWSET function.

With the BULK provider, you can import entire documents from the file system.

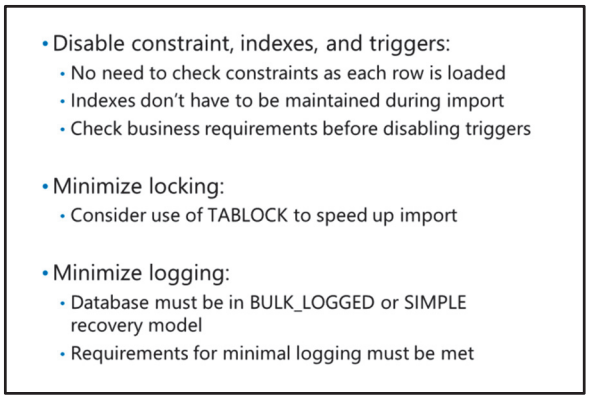
For a further overview of tools for bulk data transfer into and out of SQL Server, see the topic *Bulk Import and Export of Data (SQL Server)* in the SQL Server Technical Documentation:

Bulk Import and Export of Data (SQL Server)

https://aka.ms/Fdm7sf

Improving the Performance of Data Transfer

If you have defined constraints, indexes, and triggers on the tables that are the targets of data transfers, SQL Server checks the data values as they are imported. This checking can substantially slow down SQL Server data transfers.



Disabling Constraints, Indexes, and Triggers

Rather than checking each value during the import process or updating each index for every row, you might improve overall performance by disabling the process of checking or index updating until all the data is loaded, and then performing that work once, at the end of the import process.

For example, consider a FOREIGN KEY constraint that ensures that the relevant customer does exist whenever a customer order is inserted into the database. While you could check this reference for each customer order, it is possible that a customer may have thousands of orders, resulting in thousands of checks. Instead of checking each value as it is inserted, you can check the customer reference as a single lookup after the overall import process completes—to cover all customer orders referring to that customer.

In the way that avoiding lookups for FOREIGN KEY constraints during data import can improve performance, avoiding constant updating of indexes can have a similar effect. In many cases, rebuilding the indexes after the import process is complete is much faster than updating the indexes as the rows are imported. The exception to this situation is when there is a much larger number of rows already in the table than are being imported.

Triggers are commands that are executed when data is modified. Triggers operate on batches of rows—they do not fire once per inserted row, but instead once for each batch. It is important to decide if the processing that the triggers perform would also be better processed in bulk after the import; if you enforce business rules or initiate processes using triggers, you might not be able to disable them during bulk import without a lot of additional work when the import is complete.

Control Locking Behavior

By default, SQL Server manages the granularity of the locks it acquires during the execution of commands. SQL Server starts with row level locking and only tries to escalate when a significant number of rows are locked within a table. Managing large numbers of locks occupies resources that could be used to minimize the execution time for queries. As the data in tables that are the target of bulk-import operations are typically only accessed by the process that is importing the data, the advantage of row level locking is often not present. For this reason, it may be advisable to lock the entire table by using a TABLOCK query hint during the import process when importing data to a row-oriented table.

Bulk operations that insert data into columnstore tables do not benefit from table level locking, because each worker in a parallelized or concurrent bulk insert operation adds data to its own columnstore rowgroup.

For more information on bulk loading into columnstore tables, see the topic *Columnstore Indexes Data Loading* in the SQL Server Technical Documentation:

Columnstore Indexes Data Loading

https://aka.ms/Wu831v

Use Minimal Logging

Minimal logging is a special operation that can provide substantial performance improvements—for example, in bulk imports. In addition to making the operations faster, minimal logging helps avoid excessive log growth during large import operations for databases using the fully logged recovery model. You can implement minimal logging by changing the database recovery model from FULL to BULK_LOGGED; this is intended to be a temporary change during bulk operations. A database using the SIMPLE recovery model will use minimal logging if a bulk operation meets the necessary criteria.

Not all commands can use minimal logging. While not an exhaustive list, the items below indicate the types of restrictions that must be met to apply minimal logging:

- The table is not an article in a replication publication.
- Table locking is specified (using TABLOCK).
- If the table has no clustered index but has one or more nonclustered indexes, data pages are always minimally logged. How index pages are logged, however, depends on whether the table is empty.
- If the table is empty, index pages are minimally logged.
- If the table is nonempty, index pages are fully logged.
- If the table has a clustered index and is empty, both data and index pages are minimally logged.
- If a table has a clustered index and is nonempty, data pages and index pages are both fully logged, regardless of the recovery model.

For more information about SQL Server recovery models, see the topic *Recovery Models (SQL Server)* in the SQL Server Technical Documentation:

Recovery Models (SQL Server)

https://aka.ms/Oh7uwg

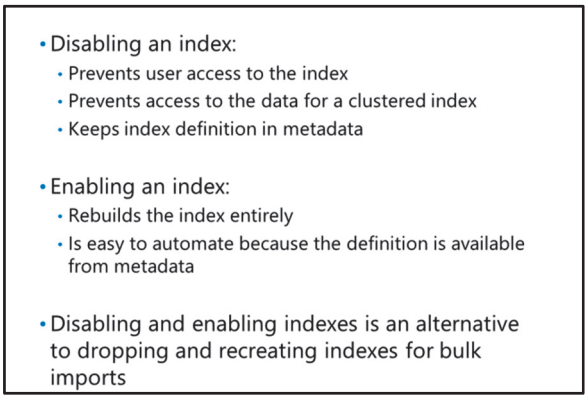
For more information about the prerequisites for minimally logged operations, see the topic *Prerequisites for Minimal Logging in Bulk Import* in the SQL Server Technical Documentation:

Prerequisites for Minimal Logging in Bulk Import

https://aka.ms/Bcktk1

Disabling and Rebuilding Indexes

To prevent an index from being accessed or updated, you can mark it as disabled. Rather than totally dropping the index details from the database, this option leaves the metadata about the index in place, but makes the index inactive. Queries that are executed by users will not use disabled indexes. You can disable an index by using the graphical interface in SQL Server Management Studio (SSMS) or by using the ALTER INDEX Transact-SQL statement.



The following code example disables an index named **idx_emailaddress** on the **dbo.Customer** table:

Disable an Index

```
ALTER INDEX idx_emailaddress ON dbo.Customer DISABLE;
```

You can disable all of the indexes on a table, as shown in the following code example:

Disable All Indexes on a Table

```
ALTER INDEX ALL ON dbo.Customer DISABLE;
```

Note: A clustered index defines how data in a table is physically ordered. If the clustered index is disabled, the table becomes unusable until the index is rebuilt.

The major advantage of disabling an index—instead of dropping it—is that you can put the index back into operation by using a rebuild operation. When you rebuild an index, you do not have to know details of the index definition. This makes it easier to create administrative scripts that stop indexes being updated while large import or update operations are taking place—and that put the indexes back into operation after those operations have completed.

For more information about disabling indexes, see the topic *Disable Indexes and Constraints* in the SQL Server Technical Documentation:

Disable Indexes and Constraints

https://aka.ms/Q677m6

Rebuilding Indexes

After data has been imported, you can enable indexes again. To enable a disabled index, you must rebuild it. You can rebuild the indexes on a table by using the graphical tools in SSMS or by using the ALTER INDEX Transact-SQL statement or the DBCC DBREINDEX command.

The following code example shows how to rebuild the **idx_emailaddress** on the **dbo.Customer** table:

Rebuild an Index

```
ALTER INDEX idx_emailaddress ON dbo.Customer REBUILD;
```

You can also use the ALL keyword with the ALTER INDEX statement to rebuild all indexes on a specified table—similar to disabling an index.

Note: In addition to enabling a disabled index, you might rebuild an index as part of regular index maintenance.
An index rebuild can be minimally logged, if the database is using the BULK_LOGGED or SIMPLE recovery models.

For more information on rebuilding indexes, see the topic *ALTER INDEX (Transact-SQL)* in the SQL Server Technical Documentation:

ALTER INDEX (Transact-SQL)

https://aka.ms/V8ygd2

Recreating Indexes

In some circumstances, the cost of rebuilding an index after it has been disabled may be greater than dropping and recreating the index. To recreate an index, you must know the index definition.

To recreate an index, replacing the existing one, you can use the CREATE INDEX statement with the DROP_EXISTING option as shown in the following example:

Recreate an Index

```
CREATE INDEX idx_emailaddress ON dbo.Customer(EmailAddress)
WITH (DROP_EXISTING);
```

Disabling and Enabling Constraints

You can use constraints to define how SQL Server enforces data integrity rules. Whenever data changes are made to a table, the new data is validated against any constraints on the table. Checking constraints during a bulk load operation can significantly increase the time taken to complete the load; it can often be more efficient to disable some or all of the constraints on a table during a bulk load operation, and then enable them after the load completes.

- Disabling PRIMARY KEY and UNIQUE constraints:
  - Achieved by disabling the associated index
  - Causes associated indexes to be rebuilt when enabled
  - Can cause failures when re-enabled if data that violates the constraint exists
  - Causes associated foreign key constraints to be disabled
- Disabling FOREIGN KEY and CHECK constraints:
  - Performed directly on the constraint
  - Constraint must be enabled WITH CHECK to verify existing data, otherwise it is untrusted

Disabling Primary Key or Unique Constraints

Primary key constraints define the column or columns that uniquely identify each row in a table. Unique constraints ensure that a column or columns do not contain duplicate values. SQL Server creates indexes to help it enforce primary key and unique constraints.

To disable a primary key or unique constraint, you must first disable the index that is associated with the constraint. When you re-enable the constraint, the associated indexes are automatically rebuilt. If duplicate values are found during the rebuild, the re-enabling of the constraint will fail. For this reason, if you disable these constraints while importing data, you must be sure that the data being imported will not violate the rules that the constraints enforce.

Note: Disabling primary key and unique constraints is only possible for nonclustered constraints. If a table has a primary key or unique constraint enforced with a clustered index, disabling the index associated with the constraint prevents access to any data in the table.

When you disable a primary key constraint, any foreign key constraints that reference the disabled primary key are also disabled.

 **Best Practice:** In general, you should not disable primary key or unique constraints during bulk load operations without very good reason. Both constraint types are critical to the integrity of your data; it is likely to be easier to prevent invalid data from being loaded—by leaving primary key and unique constraints in place—than it would be to correct it after a bulk load.

Foreign Key and Check Constraints

You use foreign key constraints to make sure that the entities in one table that are referred to by the entities in another table actually exist. For example, a supplier must exist before a purchase order can be entered. Foreign key constraints use primary key or unique constraints while checking the references. If you disable the primary key or unique constraint that a foreign key reference points to, the foreign key constraint is automatically disabled. However, when you re-enable the primary key or unique constraint, foreign key references that use these constraints are not automatically re-enabled.

You can use check constraints to limit the values that can be contained in a column or the relationship between the values in multiple columns in a table.

You can disable both foreign key and check constraints by using the NOCHECK option of the ALTER TABLE statement:

Disabling Foreign Key and Check Constraints

```
ALTER TABLE Person.Salary NOCHECK CONSTRAINT SalaryCap;
```

You can also enable foreign key and check constraints with the ALTER TABLE statement with the CHECK option. When you specify the CHECK option alone:

- The check constraint is only applied to new data added to the table after the constraint is re-enabled.
- Existing data in the table is not checked against the constraint.
- The constraint is marked as untrusted in the system metadata, and is ignored for query plan generation.

To force all existing data to be checked against the constraint, use the WITH CHECK CHECK option:

Enabling Foreign Key and Check Constraints WITH CHECK

```
ALTER TABLE Person.Salary WITH CHECK CHECK CONSTRAINT SalaryCap;
```

Using the WITH CHECK CHECK option will mark the constraint as trustworthy. For a large table, checking all existing data against a constraint can be an expensive and time-consuming operation, which might generate many thousands of logical reads.

 **Note:** The **bcp** and BULK INSERT transfer tools both ignore check and foreign key constraints by default when importing data, which causes the constraints to be marked as untrusted. Use the CHECK_CONSTRAINT option with these tools to force check and foreign key constraints to be tested for bulk-inserted data.

Demonstration: Disabling and Enabling Constraints

In this demonstration, you will see how to:

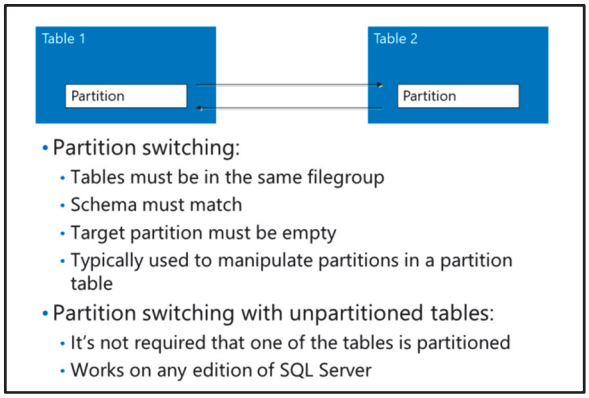
- Disable and enable primary key constraints.
- Disable and enable check constraints.
- Disable and enable foreign key constraints.
- Disable and enable unique constraints.

Demonstration Steps

1. Ensure that the **MT17B-W2016-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.
3. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod15\Demo** folder.
4. Open the query file **Demo 01 - constraints.sql**.
5. Connect the query window to your copy of the **AdventureWorksLT** database.
6. Execute the code under the heading for **Step 2** to create two tables for this demonstration.
7. Execute the code under the heading for **Step 3** to show the current state of the check constraint.
8. Execute the code under the heading for **Step 4** to disable the check constraint.
9. Execute the code under the heading for **Step 5** to show that the check constraint is marked as disabled and untrusted.
10. Execute the code under the heading for **Step 6** to enable the check constraint with CHECK.
11. Execute the code under the heading for **Step 7** to show that the constraint is enabled and marked untrusted.
12. Execute the code under the heading for **Step 8** to enable the check constraint WITH CHECK CHECK.
13. Execute the code under the heading for **Step 9** to show that the constraint is enabled and trusted.
14. Execute the code under the heading for **Step 10** to disable a nonclustered primary key.
15. Execute the code under the heading for **Step 11** to show the state of the indexes on the table.
16. Execute the code under the heading for **Step 12** to demonstrate that data can still be inserted into the table.
17. Execute the code under the heading for **Step 13** to enable the index.
18. Execute the code under the heading for **Step 14** to show the state of the indexes on the table.
19. Execute the code under the heading for **Step 15** to disable a clustered primary key constraint. Note the warning messages generated by this command.
20. Execute the code under the heading for **Step 16** to show that all the indexes on the table are disabled.
21. Execute the code under the heading for **Step 17** to enable the clustered index.
22. Execute the code under the heading for **Step 18** to show that the nonclustered index remains disabled.

23. Execute the code under the heading for **Step 19** to enable the nonclustered index.
24. Execute the code under the heading for **Step 20** to enable the foreign key constraint that references the clustered primary key.
25. Execute the code under the heading for **Step 21** to drop the demonstration objects.
26. Leave SSMS open for the next demonstration.

Partition Switching



Partition switching is a feature designed to assist with the management of data in partitioned tables— typically for bulk loading or for archiving. You can use partition switching to swap a partition in one table with a partition in another table. Because the switch operation requires only an update of metadata, it completes almost instantly, regardless of the amount of data in the partitions being switched.

To use partition switching, the source and target tables must have an identical schema—including the order of the columns in the table—and must be stored in the same filegroup. The target partition must be empty; partition switching will not work if both the source and target partitions contain data.

To switch partitions, use the ALTER TABLE… SWITCH statement. The following statement switches partition four of the partitioned table **dbo.FX_rate** with the single partition of the unpartitioned **dbo.FX_rate_staging** table:

Partition Switching

```
ALTER TABLE dbo.FX_rate_staging SWITCH TO dbo.FX_rate PARTITION 4;
```

You might use partition switching as a method of loading data into a partitioned table while maximizing the time the table is accessible to users.

Partition Switching Between Unpartitioned Tables

Partition switching can also be used to switch partitions between two unpartitioned tables—providing the tables meet the requirements for partition switching already outlined in this topic.

This means you can use partition switching on editions of SQL Server that do not support partitioning.

A partition switch with unpartitioned tables also uses ALTER TABLE…SWITCH. The following statement switches the unpartitioned table **dbo.FX_rate_2** with the unpartitioned **dbo.FX_rate_2_staging** table:

Partition Switching—Unpartitioned Tables

```
ALTER TABLE dbo.FX_rate_2_staging SWITCH TO dbo.FX_rate_2;
```

For more information on partition switching, see the topic *ALTER TABLE (Transact-SQL)* in the SQL Server Technical Documentation:

ALTER TABLE (Transact-SQL)

https://aka.ms/Tawvw0

Demonstration: Switching Partitions for Data Transfer

In this demonstration, you will see how to use partition switching to:

- Load data into a partitioned table.
- Load data into an unpartitioned table.

Demonstration Steps

1. In Solution Explorer, open the query file **Demo 02 - partition switch.sql**.
2. Connect the query window to your copy of the **AdventureWorksLT** database.
3. Execute the code under the heading for **Step 2** to create a partition function, partition scheme and partitioned table.
4. Execute the code under the heading for **Step 3** to create and add data to the unpartitioned table that matches the schema of the partitioned table.
5. Execute the code under the heading for **Step 4** to switch partition one of the partitioned table with the unpartitioned table.
6. Execute the code under the heading for **Step 5** to demonstrate the effect of the switch.
7. Execute the code under the heading for **Step 6** to create three identical unpartitioned tables, and add data to **SalesLT.ShippingRate**.
8. Execute the code under the heading for **Step 7** to add data to **SalesLT.ShippingRateStaging**, representing a data load.
9. Execute the code under the heading for **Step 8** to switch partitions. Note the use of the third table so that one of the participants in a switch is always empty.
10. Execute the code under the heading for **Step 9** to demonstrate the effect of the switch.
11. Execute the code under the heading for **Step 10** to drop the demonstration objects.
12. Leave SSMS open for the next demonstration.

Check Your Knowledge

Question	
What is the effect of disabling the clustered index on a row store table?	
Select the correct answer.	
	The index is ignored, but the table can be updated.
	The table becomes read-only.
	The table is completely inaccessible.
	The table is deleted.

Lesson 2
Importing and Exporting Table Data

This lesson begins an exploration of the tools and techniques available in SQL Server for importing and exporting data. In this lesson, you will learn about linked servers, SQL Server Integration Services, and the SQL Server Data Import and Export Wizard.

Lesson Objectives

After completing this lesson, you will be able to:

- Use linked servers.
- Work with SQL Server Integration Services.
- Use the SQL Server Data Import and Export Wizard.

Linked Servers

Linked servers provide a method for you to execute commands against remote OLE DB data sources that you access regularly from within a SQL Server instance. You can create a linked server for any data source for which a suitable OLE DB driver is installed on the operating system hosting your SQL Server instance, including other instances of SQL Server.

- Execute commands against remote data sources
- Managing Linked Servers:
  - From SSMS or using Transact-SQL
  - Define a data source and a security context for the connection
- Querying Linked Servers:
  - Four-part name:
    - server.database.schema.table
  - OPENQUERY:
    - Pass-through query—must be in linked server's query language
    - If the provider supports it, you can run DML operations against results returned by OPENQUERY

Note: You can write queries against external data providers using the OPENROWSET command, which is covered later in this module. Linked servers provide a convenient method of managing access to remote data sources at server level, rather than having them defined in Transact-SQL code.

A linked server is defined at server level, and is made up of two components:

- **An OLE DB provider**. A driver for a data source.
- **An OLE DB data source**. A specific instance of a type of data source.

The definition for a linked server data source will typically include a security context under which the connection to the remote OLE DB data source should be made. You may configure this security context in different ways, including using a single username for all connections, or mapping specific SQL Server logins to logins on the remote data source.

For more information on linked servers, see the topic *Linked Servers (Database Engine)* in the SQL Server Technical Documentation:

Linked Servers (Database Engine)

https://aka.ms/Ymesoj

Managing Linked Servers

You can create linked servers using the SSMS GUI, or by using the **sp_addlinkedserver** system stored procedure to create linked server definitions and the **sp_addlinkedserverlogin** system stored procedure to configure linked server authentication.

The details of linked server configuration vary by OLE DB provider; different OLE DB providers require—and support—different configuration options.

For more information on creating linked servers, see the topic *Create Linked Servers (SQL Server Database Engine)* in the SQL Server Technical Documentation:

Create Linked Servers (SQL Server Database Engine)

https://aka.ms/Hutpa9

For more information on creating linked servers from Transact-SQL commands, including a table of configuration values for common OLE DB providers, see the topic *sp_addlinkedserver (Transact-SQL)* in the SQL Server Technical Documentation:

sp_addlinkedserver (Transact-SQL)

https://aka.ms/Ow9493

Querying Linked Servers

There are two ways to reference linked server data in your Transact-SQL queries:

- Four-part naming.
- OPENQUERY.

Four-Part Naming

When you use four-part naming, you refer to a table or view on a linked server in the body of a Transact-SQL statement, almost as if it were part of the local database—the difference being that you qualify the object name with four parts:

- Linked server name.
- Database name (on the linked server).
- Schema name (on the linked server).
- Object name (on the linked server).

The following example demonstrates the use of a four-part name to query a table on the **Manufacturing** linked server:

Four-Part Naming

```
SELECT p.PartId, m.manufacture_date
FROM Production.Part AS p
JOIN Manufacturing.mf1.prodution.job AS m
ON m.part_identifier = p.PartId;
```

Note: When referenced using four-part names, the performance of linked server queries can be unpredictable. In some circumstances, SQL Server might attempt to apply a filter or a join to a linked server table by retrieving the entire contents of the remote table and filtering it locally. If the remote table is very large, this can be a time-consuming operation.

OPENQUERY

The OPENQUERY command is used to execute a command against a linked server as a pass-through operation, as if SQL Server were a client application of the remote data source. The command might be any valid command in the query language supported by the remote data source; if the remote data source uses a different dialect of SQL than Transact-SQL, commands issued with OPENQUERY must be in the remote data source dialect.

The following example uses OPENQUERY to return data from the **Manufacturing** linked server:

OPENQUERY SELECT

```
SELECT * FROM OPENQUERY(Manufacturing, 'SELECT TOP 10 manufacture_date FROM
mf1.prodution.job WHERE part_identifier = 1624 ORDER BY manufacture_date DESC');
```

You may also issue DML commands against the result set returned by OPENQUERY, if the OLE DB provider supports it.

The following example uses OPENQUERY to update a row in a table on the **Manufacturing** linked server:

OPENQUERY UPDATE

```
UPDATE OPENQUERY (Manufacturing, 'SELECT cancel_job FROM mf1.prodution.job WHERE id =
1208')
SET cancel_job = 'Y';
```

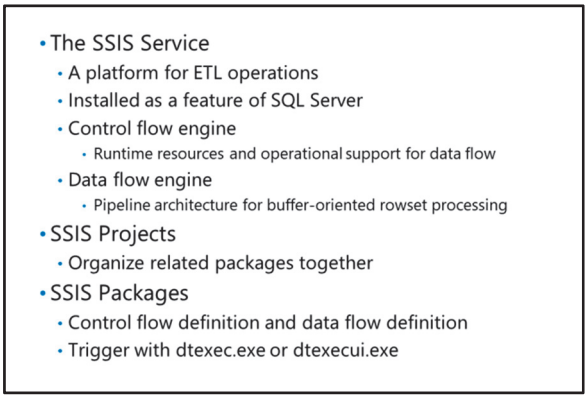
For more information on using OPENQUERY, see the topic *OPENQUERY (Transact-SQL)* in the SQL Server Technical Documentation:

OPENQUERY (Transact-SQL)

https://aka.ms/Xw7tts

Overview of SQL Server Integration Services

SSIS is an extensible platform for building complex ETL solutions. It is included with SQL Server and consists of a Microsoft Windows service that manages the execution of ETL workflows, along with tools and components for developing them.



The SSIS Service

The SSIS service is primarily a control flow engine that manages the execution of task workflows, which are defined in packages. When you are developing an SSIS package, the task workflow is referred to as the control flow. The control flow can include a special type of task to perform data flow operations. SSIS executes these tasks using a data flow engine that encapsulates the data flow in a pipeline architecture. Each step in the data flow task operates in sequence on a rowset of data as it passes through the pipeline. The data flow engine uses buffers to optimize the data flow rate, resulting in a high-performance ETL solution. In addition to the SSIS Windows service, SSIS includes:

- **SSIS Designer**. A graphical design interface for developing SSIS solutions in the Microsoft Visual Studio® development environment. Typically, you start the SQL Server Data Tools (SSDT) application to access this.

- **Wizards**. Graphical utilities you can use to quickly create, configure, and deploy SSIS solutions.
- **Command-line tools**. Utilities you can use to manage and execute SSIS packages.

An SSIS solution usually consists of one or more SSIS projects, each containing at least one SSIS package.

SSIS Projects

In SQL Server, a project is the unit of deployment for SSIS solutions. You can define project-level parameters so that users can specify run-time settings, and project-level connection managers that reference data sources and destinations used in package data flows. You can then deploy projects to an SSIS catalog in a SQL Server instance, and configure project-level parameter values and connections as appropriate for execution environments. You can use SSDT to create, debug, and deploy SSIS projects.

SSIS Packages

A project contains one or more packages, each defining a workflow of tasks to be executed. The workflow of tasks is referred to as its control flow. A package control flow can include one or more data flow tasks, each of which encapsulates its own pipeline. You can include package-level parameters so that the package receives dynamic values at run time. In previous SSIS releases, deployment was managed at the package level. In SQL Server, you can still deploy individual packages in a package deployment model.

SSIS provides two utilities that you can use to run packages:

- **DTExec utility**. You can use **DTExec** to run SSIS packages from the command line. You have to specify parameters including the server to use, the location of the package, environment variables, and input parameters. The utility reads the command-line parameters, loads the package, configures the package options based on the parameters passed, and then runs the package. It returns an exit code signifying the success or failure of the package.
- **DtExecUI utility**. The Execute Package Utility (**DtExecUI**) can run SSIS packages from SQL Server Management Studio (SSMS) or from the command prompt and is a GUI for the **DTExec** command prompt utility. The GUI simplifies the process of passing parameters to the utility and receiving exit codes.

For more information on SSIS, see the topic *SQL Server Integration Services* in the SQL Server Technical Documentation:

SQL Server Integration Services

https://aka.ms/Y0aajz

Demonstration: Working with SSIS

In this demonstration, you will see how to use SSIS to import a text file into a SQL Server database.

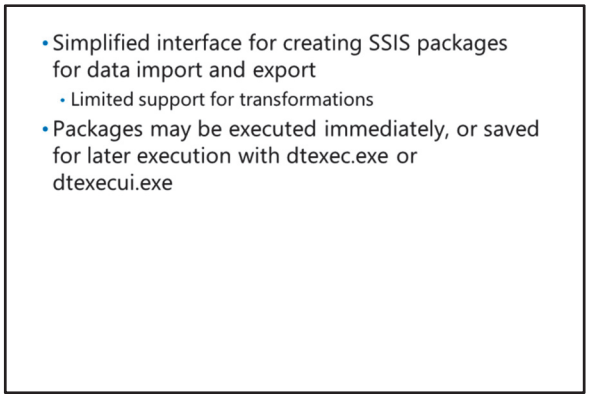
Demonstration Steps

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\Mod15** folder as Administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. Start Visual Studio, and open the **SSISProject.sln** solution in the **D:\Demofiles\Mod15\SSISProject** folder.
5. Give a brief demonstration of the different areas of an SSIS project in Visual Studio.

6. In Solution Explorer, double-click **Package.dtsx**.
7. On the **SSIS** menu, click **SSIS Toolbox**.
8. Click and drag the **Data Flow Task** from the SSIS Toolbox pane to the Package.dtsx [Design] pane. Release the **Data Flow Task** anywhere within the **Control Flow** tab of the Package.dtsx [Design] pane.
9. Right-click **Data Flow Task**, click **Rename**, type **Top Level Domain Name Import**, and then press Enter.
10. Double-click **Top Level Domain Name Import** to go to the **Data Flow** tab of the designer.
11. Click and drag **Source Assistant** from the SSIS Toolbox pane to the **Data Flow** tab of the Package.dtsx [Design] pane.
12. In the **Source Assistant - Add New Source** dialog box, in the **Select source type** box, click **Flat File**.
13. In the **Select connection managers** box, double-click **New**.
14. In the **Flat File Connection Manager Editor** dialog box, on the **General** page, in the **Connection Manager Name** box, type **TLD File**.
15. In the **File name** box, type **D:\Demofiles\Mod15\Data\top_level_domains.txt**.
16. In the **Header rows to skip** box, type **1**.
17. Clear the **Column names in the first data row** check box.
18. On the **Columns** page, examine the preview to ensure that two columns are shown.
19. On the **Advanced** page, for **Column 0**, change the **OutputColumnWidth** to **100**, and then click **OK**.
20. Right-click **Flat File Source**, click **Rename**, type **TLD File Source**, and then press Enter.
21. Click and drag **Destination Assistant** from the SSIS Toolbox pane to the **Data Flow** tab.
22. In the **Destination Assistant - Add New Destination** dialog box, confirm that **Select destination type** is **SQL Server**.
23. In the **Select Connection managers** box, click **New**, and then click **OK**.
24. In the **Connection Manager** dialog box, in the **Server name** box, type **MIA-SQL**.
25. In the **Select or enter a database name** list, click **salesapp1**, and then click **Test Connection**.
26. In the **Connection Manager** dialog box, note the test was successful, and then click **OK**.
27. In the **Connection Manager** dialog box, click **OK**.
28. Right-click **OLE DB Destination**, click **Rename**, type **salesapp1 DB**, and then press Enter.
29. Click **TLD File Source**, then click the left (blue) arrow on the bottom of the **TLD File Source** object, and then click the **salesapp1 DB** object.
30. Double-click the **salesapp1 DB** object.
31. In the **OLE DB Destination Editor** dialog box, on the **Connection Manager** page, in the **Name of the table or the view** list, click **[dbo].[TopLevelDomain]**. Point out the **Table Lock** and **Check constraints** check boxes and relate them back to the previous lesson.
32. On the **Mappings** page, in the first **Input Column** box, click **Column 0**.
33. In the second **Input Column** box, click **Column 1**, and then click **OK**.
34. On the **Debug** menu, click **Start Debugging**.
35. When the package has completed, on the **Debug** menu, click **Stop Debugging**.

36. In SQL Server Management Studio, open the query file **Demo 03 - SSIS.sql**.
37. On the **Query** menu, point to **Connection**, and then click **Change Connection**,
38. In the **Connect to Database Engine** dialog box, in **Server name** box, type **MIA-SQL**, and in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
39. Execute the query in the file to view the uploaded contents of the **dbo.TopLevelDomain** table.
40. Close Visual Studio without saving changes. Leave SSMS open for the next demonstration.

The SQL Server Import and Export Wizard



The SQL Server Import and Export Wizard offers a simplified interface for creating and executing SSIS packages; as the name suggests, it is primarily designed for use when importing and exporting data.

The Import and Export Wizard provides minimal transformation capabilities. Except for setting the name, the data type, and the data type properties of columns in new destination tables and files, the SQL Server Import and Export Wizard does not support column-level transformations during data import. You can specify a Transact-SQL query as the source for a data export. If you want to develop a more complex ETL solution, you should use SSDT to create more complex SSIS packages.

As with a full SSIS package, you can use the SQL Server Import and Export Wizard to copy data to and from any data source for which a managed .NET Framework data provider or a native OLE-DB provider is available. For example, SQL Server, flat files, Microsoft Office Access®, Microsoft Office Excel®, and a wide variety of other database engines—including a generic ODBC (Open Database Connectivity) provider that can be used to access system ODBC connections.

Note: On a 64-bit computer, SSIS setup installs the 64-bit version of the Import and Export Wizard. However, some data sources might only have 32-bit providers. To use these data sources, you must install the 32-bit version of the Data Import and Export Wizard. Selecting the **Management Tools - Complete** option during installation installs both the 32-bit and 64-bit versions of the wizard.

You can use the wizard to perform the data transfer immediately, or you can save the SSIS package it generates for execution at a later time. You can edit an SSIS package generated by the Import and Export Wizard using SSDT.

The SQL Server Import and Export Wizard can be accessed from the Windows Start menu, from SSDT, or through SSMS. You can only start the 32-bit version of the wizard from SSMS. If they are installed, the Start menu will contain links to both the 32-bit and 64-bit versions of the wizard.

For more information on the SQL Server Import and Export Wizard, see the topic *SQL Server Import and Export Wizard* in the SQL Server Technical Documentation:

SQL Server Import and Export Wizard

https://aka.ms/Xh3cbv

Demonstration: Using the SQL Server Import and Export Wizard

In this demonstration, you will see how to work with the SQL Server Import and Export Wizard.

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. In **SQL Server Management Studio**, in **Object Explorer**, click **connect**, and then click **Database Engine**.
3. In the **Connect to Database Engine** dialog box, in the **Server name** box, type **MIA-SQL**, and then click **Connect**.
4. In Object Explorer, under **MIA-SQL**, expand **Databases**, right-click **salesapp1**, point to **Tasks**, and then click **Export Data**.
5. In the **SQL Server Import and Export Wizard** window, click **Next**.
6. On the **Choose a Data Source** page, in the **Data source** box, click **SQL Server Native Client 11.0**.
7. Verify that the **Database** box has the value **salesapp1**, and then click **Next**.
8. On the **Choose a Destination** page, in the **Destination** box, click **Flat File Destination**.
9. In the **File name** box, type **D:\Demofiles\Mod15\export.txt**, and then click **Next**.
10. On the **Specify Table Copy or Query** page, verify that **Copy data from one or more tables or views** is selected, and then click **Next**.
11. On the **Configure Flat File Destination** page, in the **Source table or view** list, click **[Production].[Categories]**, and then click **Next**.
12. On the **Save and Run Package** page, verify that **Run immediately** is selected, and then click **Finish**.
13. On the **Complete the Wizard** page, click **Finish** to run the export.
14. When the export completes, click **Close**.
15. Using File Explorer, open **D:\Demofiles\Mod15\export.txt** to verify the result of the export.
16. Close Notepad.
17. Leave SSMS open for the next demonstration.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? The Data Import and Export Wizard can only be used to import and export data to Microsoft formats (such as Excel, Access, and SQL Server).	

Lesson 3
Using bcp and BULK INSERT to Import Data

This lesson continues an exploration of the tools and techniques available in SQL Server for importing and exporting data. In this lesson, you will learn about **bcp**, BULK INSERT, and OPENROWSET.

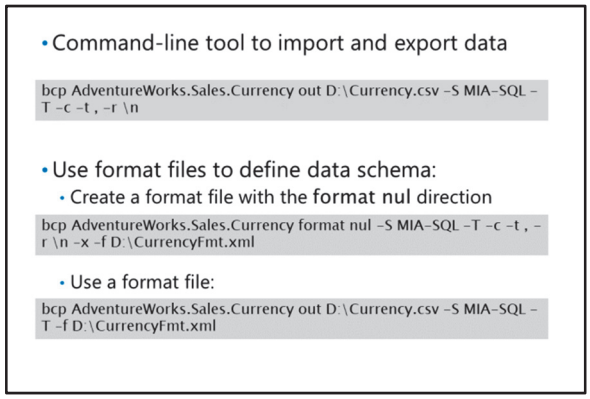
Lesson Objectives

After completing this lesson, you will be able to:

- Work with **bcp** to import and export data.
- Use the BULK INSERT command to import data.
- Work with the OPENROWSET function to import data.

The bcp Utility

You can use the **bcp** command-line utility to bulk copy data between an instance of Microsoft SQL Server and a data file in a user-specified format. You can use it to easily import large numbers of new rows into SQL Server tables or to export data out of tables into data files.



bcp Syntax

The syntax for the **bcp** utility is versatile, and includes a large number of options. The general form of a **bcp** command specifies:

- A table or view in a SQL Server database, which will be the source for data export or the target for data import.
- A direction (in when importing data into SQL Server; out when exporting data from SQL Server).
- A local file name for the source (when importing) or destination (when exporting).

You can also use the **queryout** direction to specify that data is to be extracted from the database based on a Transact-SQL query. Additionally, the **bcp** utility supports the following commonly-used arguments. Note that the arguments are case-sensitive:

- **-S *server\instance***. Specifies the SQL Server instance. The default is (local).
- **-d**. The database containing the table or view (you can also specify a fully-qualified table or view name that includes the database and schema—for example, **AdventureWorks.Sales.Currency**).
- **-T**. Specifies that a trusted connection should be used to connect using Windows authentication.
- **-U *user_name***. Specifies a user name for SQL Server authentication.
- **-P *password***. Specifies a password for SQL Server authentication.
- **-c**. Specifies that the data file stores data in character format.
- **-w**. Specifies that the data file stores data in wide (Unicode) character format.
- **-n**. Specifies that the data file stores data in SQL Server native format.
- **-f *format_file***. Specifies a format file that defines the schema for the data.

- **-t *delimiter***: Specifies a field terminator for data in character format. The default is a tab.
- **-r *delimiter***: Specifies a row terminator for data in character format. The default is a new line.
- **-?**. Displays a complete list of switches.

 **Note:** By default, **bcp** will use column default values when inserting to columns for which no value is specified. Use the **-k** argument to override this behavior and insert NULL instead of using defaults.

For more information in handling NULL in **bcp** operations, see the topic *Keep Nulls or Use Default Values During Bulk Import (SQL Server)* in the SQL Server Technical Documentation:

 **Keep Nulls or Use Default Values During Bulk Import (SQL Server)**

https://aka.ms/O2popc

Character Formats and Field and Row Terminators

If you are using **bcp** to transfer data to or from a non-SQL Server system, you have to use the **-c** argument if the file contains single-byte character data, or the **-w** argument if the file contains multi-byte (Unicode) character data. When you use **-c** or **-w** to import or export data, you should avoid using the default field delimiter (a tab character) and row delimiter (a new line character) if there is any chance that these characters might appear in the data.

If you are using **bcp** to transfer data between instances of SQL Server, you should use the **-n** argument to specify native formatting. In native format, **bcp** files occupy less space than equivalent character or wide character files; using the native format means that you do not have to specify field and row delimiters.

The following example connects to the **MIA-SQL** SQL Server instance using Windows authentication, and exports the contents of the **Sales.Currency** table in the **AdventureWorks** database to a text file named **Currency.csv**, in which the data is saved in comma-delimited character format with a new line for each row:

Using bcp to Export Data

```
bcp AdventureWorks.Sales.Currency out D:\Currency.csv -S MIA-SQL -T -c -t , -r \n
```

Using Format Files

While you can export and import data easily by using delimited character data or SQL Server native format, there are some scenarios where you might want to use specific data formats for the data you are importing or exporting. If you run **bcp** using the **-c** or **-w** arguments without specifying any format information, the utility will prompt you to specify the data type, prefix length, and delimiter for each field in the specified table or view. The utility will also give you the option to save the specified schema as a format file that you can reuse in a later **bcp** operation. Format files define the schema of the data for your import/export operations, and can be defined as text or XML files.

To pre-emptively create a format file, use the **format nul** direction and specify the name of the format file you want to create. You can then interactively specify the data type, prefix length, and delimiter for each field in the specified table or view, and save the resulting schema in the format file. The default format file type is text, but you can use the **-x** argument to create an XML format file. If you want to create a format file for character data with specific field and row terminators, you can specify them with the **-c**, **-t**, and **-r** argument.

The following example shows how to use **bcp** to create an XML-based format file named **CurrencyFmt.xml** based on the **AdventureWorks.Sales.Currency** table:

Create an XML Format File

```
bcp AdventureWorks.Sales.Currency format nul -S MIA-SQL -T -c -t , -r \n -x -f
D:\CurrencyFmt.xml
```

To use a format file when importing or exporting data, use the **-f** argument.

The following example shows how to import the contents of **Currency.csv** into the **Finance.dbo.Currency** table. The **in** parameter specifies the file to read and the **-f** argument specifies the format file to use:

Import Data with a Format File

```
bcp Finance.dbo.Currency in D:\Currency.csv -S MIA-SQL -T -f D:\CurrencyFmt.xml
```

 **Note:** When you use a format file to import data, empty string values in the source file will be converted to NULL.

For more information on **bcp**, including a full description of accepted arguments, see the topic *bcp Utility* in the SQL Server Technical Documentation:

 **bcp Utility**

https://aka.ms/Ewe8lb

Demonstration: Working with bcp

In this demonstration, you will see how to work with the **bcp** utility to:

- Create a format file.
- Export data.

Demonstration Steps

1. Open a command prompt, type the following command, and then press Enter to view the **bcp** syntax help:

   ```
   bcp -?
   ```

2. At the command prompt, type the following command, and then press Enter to create a text format file:

   ```
   bcp salesapp1.HR.Employees format nul -S MIA-SQL -T -w -t ^| -r \n -f
   D:\Demofiles\Mod15\bcp\EmployeesFmt.txt
   ```

3. At the command prompt, type the following command, and then press Enter to create an XML format file:

   ```
   bcp salesapp1.HR.Employees format nul -S MIA-SQL -T -w -t ^| -r \n -x -f
   D:\Demofiles\Mod15\bcp\EmployeesFmt.xml
   ```

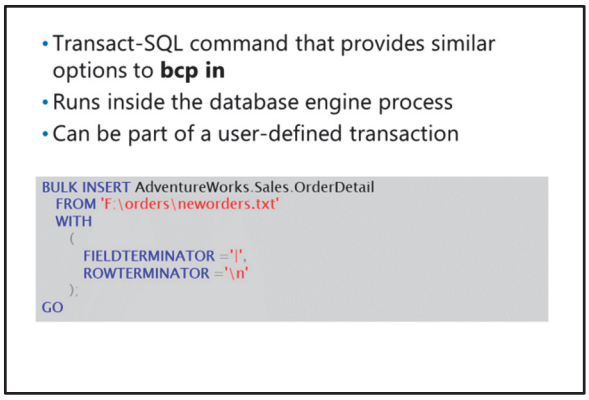
4. Using Notepad, open **D:\Demofiles\Mod15\bcp\EmployeesFmt.txt** and **D:\Demofiles\Mod15\bcp\EmployeesFmt.xml**. Review and compare the contents of the files, then close Notepad.
5. At the command prompt, type the following command, and then press Enter to export data using the XML format file:

```
bcp salesapp1.HR.Employees out D:\Demofiles\Mod15\bcp\Employees.csv -S MIA-SQL -T -f
D:\Demofiles\Mod15\bcp\EmployeesFmt.xml
```

6. Close the command prompt.
7. Using Notepad, open the **D:\Demofiles\Mod15\bcp\Employees.csv** file and view the data that has been exported. Note that the commas in several of the data fields make this data unsuitable for export using a comma as a field delimiter. Close Notepad when you have finished reviewing.

The BULK INSERT Statement

The BULK INSERT statement loads data from a data file into a table. This functionality is similar to that provided by the **in** direction of the **bcp** command. However, the data file is read by the SQL Server process, not by an external utility. The BULK INSERT statement executes within a Transact-SQL batch. Because the data files are opened by a SQL Server process, data is not copied between client process and SQL Server processes. By comparison, the **bcp** utility runs in a separate process, which can produce a higher load on the server when run on the same system.



A key consideration for using the BULK INSERT statement is that file paths to source data must be accessible from the server where the SQL Server instance is running, and must use the correct drive letters for volumes as they are defined on the server. For example, when running a BULK INSERT statement from a client computer, the path **C:\data\file.txt** references a file on the C: volume of the server, not the client.

Unlike **bcp**, you can execute the BULK INSERT statement from within a transaction you control, which gives the ability to group BULK INSERT with other operations in a single transaction. However, you should take care to ensure that the size of the data batches that you import within a single transaction are not excessive or significant log file growth might occur—even when the database is in simple recovery mode.

In the following example, new orders are inserted into the **Sales.OrderDetail** table from a text file on the file system:

Using the BULK INSERT Statement

```
BULK INSERT AdventureWorks.Sales.OrderDetail
   FROM 'F:\orders\neworders.txt'
   WITH
      (
         FIELDTERMINATOR ='|',
         ROWTERMINATOR ='\n'
      );
GO
```

Like **bcp**, BULK INSERT supports the use of format files to define the data types of the source data file.

Note: The BULK INSERT statement is not supported on Azure SQL Database. Use SSIS or **bcp** to bulk-load data into Azure SQL Database.

For more information on the BULK INSERT statement, see the topic *BULK INSERT (Transact-SQL)* in the SQL Server Technical Documentation:

BULK INSERT (Transact-SQL)

https://aka.ms/F91vki

Demonstration: Working with BULK INSERT

In this demonstration, you will see how to import data with the BULK INSERT statement.

Demonstration Steps

1. In SQL Server Management Studio, open the query file **Demo 06 - BULK INSERT.sql**.
2. Execute the code under the heading for **Step 1** to demonstrate that the **Finance.dbo.Currency** table is empty.
3. Execute the code under the heading for **Step 2** to run a BULK INSERT statement to load **Finance.dbo.Currency** with data.
4. Execute the code under the heading for **Step 3** to verify that the table has been loaded with data.
5. Leave SSMS open for the next demonstration.

The OPENROWSET Function

You can use the OPENROWSET function to access data using an OLE-DB provider, and use Transact-SQL queries to retrieve data from a wide range of external data sources. SQL Server includes a special OLE-DB provider called BULK with which you can access data in files. The same format files that are used with **bcp** and BULK INSERT can also be used with this provider.

- SELECT rows from a data file based on a format file
  - Data may then be used in other Transact-SQL statements
- Import rows from any OLE DB provider
  - **ad hoc distributed queries** server level setting must be enabled
  - OLE DB provider must be configured to allow ad hoc access
- Import a file as a BLOB into a single column/row

The following example code inserts the contents of the **Accounts.csv** text file into the **dbo.Accounts** table, using the **AccountsFmt.xml** format file to determine the schema of the rows in the text file:

Using OPENROWSET to Insert Rows from a Text File

```
INSERT dbo.Accounts
SELECT *
FROM OPENROWSET ( BULK 'D:\Accounts.csv',
                                    FORMATFILE = 'D:\AccountsFmt.xml'
                                  ) AS rows;
GO
```

Note: The result set returned by OPENROWSET must be given an alias in the FROM clause, using the AS keyword. In the previous example, the alias is **rows**.

As with the BULK INSERT statement, file paths used with the OPENROWSET function refer to volumes that are defined on the server.

Two key advantages of OPENROWSET when compared to **bcp** or BULK INSERT are that:

- OPENROWSET can be used in a query with a WHERE clause (to filter the rows that are loaded).
- OPENROWSET can be used in a SELECT statement that is not necessarily associated with an INSERT statement.

Inserting BLOB Data with OPENROWSET

In addition to the import of data rows, the BULK provider offers three special options to import the entire file contents as a binary large object (BLOB) into a single column of a table. These special options are:

- SINGLE_CLOB. This option reads an entire single-byte character-based file as a single value of data type **varchar(max)**.
- SINGLE_NCLOB. This option reads an entire double-byte character-based file as a single value of data type **nvarchar(max)**.
- SINGLE_BLOB. This option reads an entire binary file as a single value of data type **varbinary(max)**.

In the following example, the data in the **SignedAccounts.pdf** file is inserted into the **Document** column of the **dbo.AccountsDocuments** table:

Inserting Large Object Data with the OPENROWSET Function

```
INSERT INTO dbo.AccountsDocuments(FiscalYear, Document)
SELECT 2015 AS FiscalYear, *
FROM OPENROWSET(BULK 'D:\SignedAccounts.pdf', SINGLE_BLOB) AS Document;
GO
```

Note: To use OPENROWSET with OLE-DB providers other than BULK, the **ad hoc distributed queries** system configuration option must be enabled.
The OLE DB provider must also be configured to provide ad hoc access—this is normally enabled by default, but may be disabled by an administrator. This provider setting can be configured though the properties of the provider under **Providers** node, under **Server Objects**, in the SSMS Object Explorer. The **Disallow adhoc access** setting must not be selected to allow the provider to be used in ad hoc queries. Alternatively, you can use the **sp_MSset_oledb_prop** system stored procedure to set the value of the **DisallowAdHocAccess** provider setting.

Note: OPENROWSET is not supported on Azure SQL Database. Use SSIS or **bcp** to bulk-load data into Azure SQL Database.

For more information on using OPENROWSET, see the topic *OPENROWSET (Transact-SQL)* in the SQL Server Technical Documentation:

OPENROWSET (Transact-SQL)

https://aka.ms/Xmtpzp

Demonstration: Working with OPENROWSET

In this demonstration, you will see how to import data using the OPENROWSET function.

Demonstration Steps

1. In SQL Server Management Studio, open the query file **Demo 07 - OPENROWSET.sql**.
2. Execute the code under the heading for **Step 1** to demonstrate that the **Finance.dbo.SalesTaxRate** table is empty.
3. Execute the code under the heading for **Step 2** to demonstrate a SELECT statement using the OPENROWSET BULK provider.
4. Execute the code under the heading for **Step 3** to demonstrate that the output of an OPENROWSET statement can be filtered with a WHERE clause.
5. Execute the code under the heading for **Step 4** to use an OPENROWSET statement to insert data into the **Finance.dbo.SalesTaxRate** table.
6. Execute the code under the heading for **Step 5** to demonstrate that the **Finance.dbo.SalesTaxRate** table now contains data.
7. Leave SSMS open for the next demonstration.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? By default, **bcp** and BULK INSERT ignore check constraints, foreign key constraints, and triggers when importing data.	

Lesson 4
Deploying and Upgrading Data-Tier Applications

Data-tier applications (DACs) provide a way to simplify the development, deployment, and management of database applications and their SQL Server instance-level dependencies. They provide a logical container for application databases for use in installation and upgrade.

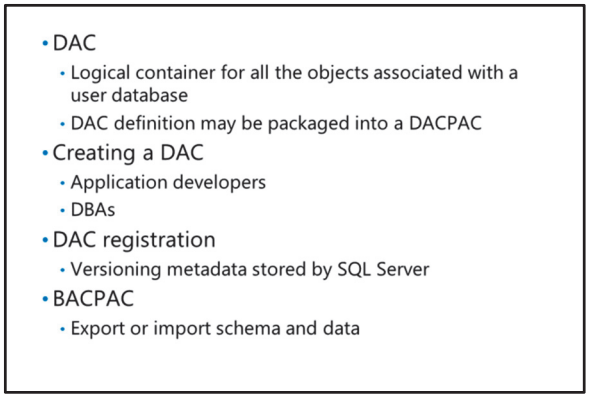
Lesson Objectives

After completing this lesson, you will be able to:

- Provide an overview of DAC.
- Deploy a DAC.
- Perform an in-place upgrade of a DAC.
- Extract a DAC from an existing database.

Data-Tier Application Overview

A DAC is a logical collection of all the database objects—such as tables, stored procedures, logins, and users—associated with a SQL Server user database. Definitions for all the objects in a DAC can be combined into a deployment package file, called a DAC package (DACPAC), which can be used to install or upgrade a DAC across multiple instances of the database engine.



DACs are intended to simplify the installation and upgrade of smaller database applications; installation and upgrade of DACs is automated because they are not designed for a large line of business applications. The intention is to make it easy to install and upgrade large numbers of simpler applications.

You can carry out the following operations on a DAC:

- **Extract**. Extract a database into a DACPAC.
- **Deploy**. Deploy a DACPAC to a host server.
- **Register**. Explicitly register a database as a DAC.
- **Unregister**. Unregister a previously registered DAC.
- **Upgrade**. Upgrade an existing database using a DACPAC.

Creating a DAC

There are two ways that a DAC might typically be created:

- **By application developers**. Developers can define a DAC as part of the application development process, using Visual Studio SSDT to create database object definitions required for their application. When development is complete, the DAC is passed to administrators for deployment to production SQL Server instances. As application requirements change and new features are added, the DAC is

updated and new DACPACs are generated so that updates can be deployed to production database instances.

- **By database administrators**. Database administrators (DBAs) can extract a DAC from an existing SQL Server database. The resulting DACPAC might be used to deploy new copies of an application database, or to deploy an application database to Azure SQL Database.

DACPACs may optionally include policies that indicate restrictions on where they can be deployed—for instance, restricting deployment to a specific version and edition of SQL Server.

 **Note:** Data-tier applications do not support all SQL Server objects. For example, XML schema collections and SQL CLR based objects are not supported. For this reason, not all databases are available for extraction to a DACPAC file. When SQL Server is unable to perform a registration or extraction, the wizard displays the objects that are not supported.

For more information about the object types supported by DACs, see the topic *DAC Support For SQL Server Objects and Versions* in the SQL Server Technical Documentation:

 **DAC Support For SQL Server Objects and Versions**

https://aka.ms/Ysmo0p

DAC Registration

When a database is deployed or upgraded from a DACPAC, the database is registered as a DAC with SQL Server. When registration occurs, the DAC version and other associated metadata is recorded by the database engine instance. DACs can be explicitly registered and unregistered using the **REGISTER** and **UNREGISTER** actions of the DAC tools.

One of the benefits of using a DAC is that the DAC is versioned and can be upgraded across the enterprise in a consistent, managed way. This is useful in large organizations where the same database application might be deployed in multiple sites or virtual servers. Application administrators can easily track which version of an application is installed in each location, and upgrade it to the latest version as required.

BACPAC

While a DACPAC contains the definition for a database schema, it is also possible to export both the schema and data of a DAC into a file format called BACPAC. BACPAC extends the DACPAC format to include scripts to recreate a database, including its data.

A BACPAC might be used to migrate an application database to Azure SQL Database.

You can carry out two operations on a BACPAC:

- **Export**. Generate a BACPAC from an existing database.
- **Import**. Use a BACPAC to create a new database.

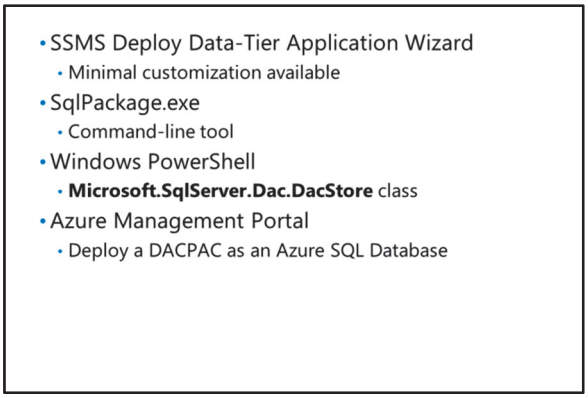
For more information about DACs, see the topic *Data-tier Applications* in the SQL Server Technical Documentation:

 **Data-tier Applications**

https://aka.ms/Hpvzso

Deploying Data-Tier Applications

A DAC can be deployed from a DACPAC using any of four tools:



- **SSMS Deploy Data-Tier Application Wizard**. With the deployment wizard, you can apply a DACPAC to a database engine instance. The wizard provides some limited customization of the database to be created from the DAC—for example, the database name, and location of data files.
- **SqlPackage.exe**. **SqlPackage**. A command-line utility capable of accomplishing many tasks related to DAC administration, including DACPAC deployment. **SqlPackage** supports a large number of arguments which offer fine control over the behavior of a DAC deployment.
- **Windows PowerShell®**. For more complex automation than can be achieved using **SqlPackage**, DACs can be deployed using PowerShell. DAC operations are carried out using the **Microsoft.SqlServer.Dac.DacStore** class.
- **Azure Management Portal**. When creating a new Azure SQL Database from the Azure Portal, you can specify a DACPAC as the source.

For more information on deploying DACs, see the topic *Deploy a Data-tier Application* in the SQL Server Technical Documentation:

Deploy a Data-tier Application

https://aka.ms/T03qx1

For further information on the **SqlPackage** utility, including a complete list of arguments, see the topic *SqlPackage.exe* on MSDN:

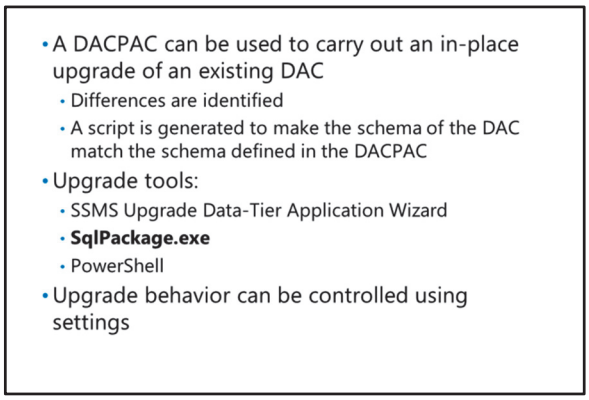
SqlPackage.exe

http://aka.ms/gaq8sj

Performing In-Place Upgrades of Data-Tier Applications

You can use a DACPAC to upgrade a DAC in-place. When you do this, the current database schema is automatically compared to the schema defined in the DACPAC; a script is generated that alters the existing schema so that it matches the schema defined in the DACPAC—then this script is applied to the DAC.

When you carry out an in-place upgrade, the application name contained in the DAC metadata must match the application name in the new DACPAC.



A DAC may be upgraded using a DACPAC, in conjunction with any of these three tools:

- **SSMS Upgrade Data-Tier Application Wizard**. With the upgrade wizard, you can apply a DACPAC to an existing DAC. The wizard can facilitate some limited customization of the database to be created from the DAC—for example, the database name, and location of data files.
- **SqlPackage.exe**. **SqlPackage** is a command-line utility capable of accomplishing many tasks related to DAC administration, including DACPAC upgrade. **SqlPackage** supports a large number of arguments that offer fine control over the behavior of a DAC upgrade.
- **PowerShell**. For more complex automation than can be achieved using **SqlPackage**, DACs can be upgraded using PowerShell.

The behavior of an in-place DAC upgrade may be configured using the following settings:

- **Ignore Data Loss**. The upgrade will proceed even if data will be lost—for example, if a table is dropped. By default, this setting is **True**.
- **Block on Changes**. The upgrade will stop if the database schema would be changed by the upgrade. By default, this setting is **False**.
- **Rollback on Failure**. The upgrade is encapsulated in a transaction, which will be rolled back in the event of an error. By default, this setting is **False**. When this setting is **False**, if an error occurs during an in-place upgrade, the only way to restore the database to a known state will be to restore a backup.
- **Skip Policy Validation**. The DACPAC validation settings are bypassed. By default, this setting is **False**.

 **Best Practice:** You should take a full database backup before proceeding with an in-place upgrade of a DAC.

Each of the three deployment mechanisms—the upgrade wizard, **SqlPackage**, and PowerShell—can be run in a mode whereby you can review the changes needed to complete the upgrade before they are applied.

For more information on upgrading a DAC in-place, see the topic *Upgrade a Data-tier Application* in the SQL Server Technical Documentation:

 **Upgrade a Data-tier Application**

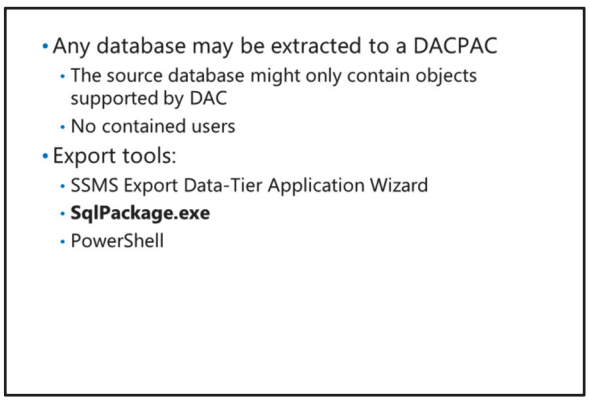
https://aka.ms/Ap3wty

Extracting Data-Tier Applications

You can generate a DACPAC from any existing database that meets the following requirements:

- The database may not contain any objects that are not supported by DAC.
- The database may not have any contained users—these are users without a login with security credentials stored at database level.

It is not a requirement that the source database be registered as a DAC.



Note: Remember that you *extract* a database when you create a DACPAC for its schema. You *export* a database when you create a BACPAC from its schema and data. Although a DACPAC does not contain data for the complete database, you may optionally include up to 10 MB of reference data in a DACPAC.

A DAC may be extracted using a DACPAC with any of these three tools:

- **SSMS Extract Data-Tier Application Wizard**. With the extraction wizard, you can create a DACPAC from an existing database. The wizard can facilitate some limited customization of the DACPAC—for example, the application name and version number.
- **SqlPackage.exe**. **SqlPackage**. A command-line utility capable of accomplishing many tasks related to DAC administration, including extracting a database to a DACPAC. **SqlPackage** supports a large number of arguments that offer fine control over the behavior of a DAC extract.
- **PowerShell**. For more complex automation than can be achieved using **SqlPackage**, DACPACs can be extracted using PowerShell.

For more information on extracting a DACPAC from a database, see the topic *Extract a DAC From a Database* in the SQL Server Technical Documentation:

Extract a DAC From a Database

https://aka.ms/Fqoemt

Demonstration: Working with Data-Tier Applications

In this demonstration, you will see how to use SSMS wizards to:

- Export a data-tier application.
- Import a data-tier application.

Demonstration Steps

1. In SQL Server Management Studio, in Object Explorer, under **MIA-SQL**, expand **Databases**, right-click **Finance**, point to **Tasks**, and then click **Extract Data-tier Application**.
2. In the **Extract Data-tier Application** dialog box, click **Next**.
3. On the **Set Properties** page, in the **Save to DAC package file (include .dacpac extension with the file name)** box, type **D:\Demofiles\Mod15\dacpac\Finance.dacpac**, and then click **Next**.
4. On the **Validation and Summary** page, click **Next**. The extract will begin.
5. On the **Build Package** page, when the extraction process is complete, click **Finish**.
6. In File Explorer, navigate to **D:\Demofiles\Mod15\dacpac** to view the exported DACPAC file.
7. To import the DACPAC, in SSMS, in Object Explorer, under **MIA-SQL**, right-click **Databases**, and then click **Deploy Data-tier Application**.
8. In the **Deploy Data-tier Application** window, click **Next**.
9. On the **Select Package** page, in the **DAC package (file name with the .dacpac extension)** box, type **D:\Demofiles\Mod15\dacpac\Finance.dacpac**, and then click **Next**.
10. On the **Update Configuration** page, in the **Name (the name of the deployed DAC and database)** box, type **FinanceDAC**, and then click **Next**.
11. On the **Summary** page click **Next**. The deployment will run.

12. On the **Deploy DAC** page, when the deployment is complete, click **Finish**.
13. In Object Explorer, right-click **Databases**, and then click **Refresh**. Verify that the **FinanceDAC** database exists.
14. Expand **FinanceDAC**, expand **Tables**, right-click **dbo.Currency**, and then click **Select Top 1000 Rows** to verify that the table has been created with no data.
15. Close SSMS, without saving any changes.

Check Your Knowledge

Question	
Which of the following is *not* an action you can carry out on a DACPAC?	
Select the correct answer.	
	EXTRACT
	UPGRADE
	DEPLOY
	REGISTER
	EXPORT

Lab: Importing and Exporting Data

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler and retailer of cycle products.

The company receives updates of currencies and exchange rates from an external provider. One of these files is provided as an Excel spreadsheet, the other file is provided as a delimited text file. You must import both these files into tables that will be used by the Accounting team.

Periodically, the Marketing team requires a list of prospects that have not been contacted within the last month. You must create and test a package that will extract this information to a file for them.

The Accounting team has purchased a new application for tracking fixed assets. The database for this application is installed as a data-tier application. You will install the DACPAC provided by the application developers.

Objectives

After completing this lab, you will be able to:

- Use the Data Import and Export Wizard.
- Import data using **bcp**.
- Import data using BULK INSERT.
- Create and test an SSIS package to extract data.
- Deploy a data-tier application.

Estimated Time: 90 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Import Excel Data Using the Import Wizard

Scenario

The Accounts department has received a list of unique currency codes in an Excel spreadsheet. You will import the data from this spreadsheet into the **AdventureWorks.Accounts.CurrencyCode** table using the Import and Export Wizard.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Examine the Excel File

3. Run the Data Import Wizard

▶ Task 1: Prepare the Lab Environment

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:\Labfiles\Lab15\Starter** folder as Administrator.

Enable Excel data import

- Install the Microsoft Access Database Engine 2016 Redistributable to enable SQL server to import Excel and Access data.

Microsoft Access Database Engine 2016 Redistributable download

https://aka.ms/Tjh400

▶ Task 2: Examine the Excel File

- Open the **currency_codes.csv** file in **D:\Labfiles\Lab15\Starter\Import**. Review the contents, then close the file when you have finished your review.

▶ Task 3: Run the Data Import Wizard

1. From the **Start** menu, start **SQL Server 2017 Import and Export Data (64-bit)**.
2. Use the **SQL Server Import and Export Wizard** to load the data from **D:\Labfiles\Lab15\Starter\Import\currency_codes.xlsx** into **AdventureWorks.Accounts.CurrencyCode**.
3. When the import is complete, open SSMS, and then open the project file **D:\Labfiles\Lab15\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - currency codes.sql**.
4. Execute the code under the heading for task 1 to view the data in the **Accounts.CurrencyCode** table.

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

Use the Import and Export Wizard to import data.

Exercise 2: Import a Delimited Text File Using bcp

Scenario

The Accounts department has received a comma-delimited text file containing historical day-end currency exchange rates. You will load this file into the **AdventureWorks.Accounts.ExchangeRate** table using **bcp**.

Note: The currency exchange rates used in this exercise are example data; they do not correspond to the actual currency exchange rates on the dates specified.

The main tasks for this exercise are as follows:

1. Import Data Using bcp
2. Trusting Foreign Key Constraints

▶ Task 1: Import Data Using bcp

- Open a command prompt and write a **bcp** command that imports the **currency_exchange_rates.txt** file in **D:\Labfiles\Lab15\Starter\Import** to the **AdventureWorks.Accounts.ExchangeRate** table. The field delimiter is a comma, and the row delimiter is a new line character.

▶ Task 2: Trusting Foreign Key Constraints

1. In SSMS Solution Explorer, open the **Lab Exercise 02 - bcp.sql** query file. Execute the query under the heading for **Task 1** to examine the status of the constraints on the **Accounts.ExchangeRate** table. Are the foreign key constraints trusted?
2. Execute the query under the heading for **Task 2** to permit the foreign key constraints to be trusted.
3. Execute the query under the heading for **Task 1** to examine the status of the constraints on the **Accounts.ExchangeRate** table again. Are the foreign key constraints trusted?

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

Import a delimited text file using **bcp**.

Exercise 3: Import a Delimited Text File Using BULK INSERT

Scenario

New copies of the currency exchange rate file will be delivered to the accounting team every month; the updated files will be uploaded by replacing the contents of the **AdventureWorks.Accounts.ExchangeRate** table with the contents of the new file. So that this exercise can be completed from within SQL Server, you will write a T-SQL script that loads the table using BULK INSERT.

The main tasks for this exercise are as follows:

1. Clear Down the Accounts.ExchangeRate Table
2. Import Data Using BULK INSERT
3. Check Foreign Key Trust Status

▶ Task 1: Clear Down the Accounts.ExchangeRate Table

- In SSMS Solution Explorer, open the **Lab Exercise 03 - BULK INSERT.sql** query file. Execute the query under the heading for **Task 1** to clear down the **Accounts.ExchangeRate** table.

▶ Task 2: Import Data Using BULK INSERT

- Under the heading for **Task 2**, edit the query to insert the contents of the **currency_exchange_rates.txt** file in **D:\Labfiles\Lab15\Starter\Import** to **Accounts.ExchangeRate** using BULK INSERT. Include an option to permit constraints to be checked during the import.

▶ Task 3: Check Foreign Key Trust Status

- Execute the query under the heading for task 3 to examine the status of the constraints on the **Accounts.ExchangeRate** table. If you correctly specified the CHECK_CONSTRAINT option in the previous task, the constraints will be marked as trusted (**is_not_trusted** = 0).

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

Import a delimited text file using BULK INSERT.

Exercise 4: Create and Test an SSIS Package to Extract Data

Scenario

Once a month, the marketing team requests a list of prospects that have not been contacted within the last 30 days. The list is generated from the output of a stored procedure—**Sales.usp_prospect_list**—in the **AdventureWorks** database. You will create an SSIS package to enable this task to be automated.

The main tasks for this exercise are as follows:

1. Create an SSIS Package

2. Add a Data Source

3. Add a File Target

4. Test the Package

▶ Task 1: Create an SSIS Package

1. Start Visual Studio® 2015, then open the solution **prospect_export.sln** in the **D:\Labfiles\Lab15\Starter\prospect_export** directory.
2. Create a new SSIS package called **prospects.dtsx**.

▶ Task 2: Add a Data Source

- Add a data source to the **prospects.dtsx** package to export the result set generated by the Sales.usp_prospect_list stored procedure in the **AdventureWorks** database.

▶ Task 3: Add a File Target

- Add a data destination to **prospects.dtsx** to route the output of the data source to a text file called **prospects.txt** in the D:\Labfiles\Lab15\Starter\Export directory.

▶ Task 4: Test the Package

1. Execute the **prospects.dtsx** package in debug mode from Visual Studio.
2. Verify that a **prospects.txt** file has been created in the **D:\Labfiles\Lab15\Starter\Export** directory.

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

Create and test an SSIS package.

Exercise 5: Deploy a Data-Tier Application

Scenario

The Accounting team has purchased a new application for tracking fixed assets. The database for this application is installed as a data-tier application. You will install the DACPAC provided by the application developers on the **MIA-SQL** instance.

The main tasks for this exercise are as follows:

1. Review the Contents of a DACPAC

2. Deploy the Data-Tier Application

▶ Task 1: Review the Contents of a DACPAC

1. Unpack the contents of the **FixedAssets_1.0.9.1.dacpac** file in **D:\Labfiles\Lab15\Starter\FixedAssets** to **D:\Labfiles\Lab15\Starter\FixedAssets\FixedAssets_1.0.9.1**.
2. Review the **model.sql** script in **D:\Labfiles\Lab15\Starter\FixedAssets\FixedAssets_1.0.9.1**.

▶ Task 2: Deploy the Data-Tier Application

- Deploy the **FixedAssets_1.0.9.1.dacpac** file in **D:\Labfiles\Lab15\Starter\FixedAssets** to the **MIA-SQL** instance using the SSMS Deploy Data-Tier Application Wizard.

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

Review the contents of a DACPAC.

Deploy a data-tier application.

Question: What alternative methods to an SSIS package might you use to export the output of the **Sales.usp_prospect_list** stored procedure to a file?

Check Your Knowledge

Question	
If the HR.JobCandidate table has included a column for a resumé in Microsoft Word document format, which of the following commands could you use to import the document into a column in a table?	
Select the correct answer.	
	The BULK provider in the OPENROWSET command with the SINGLE_BLOB option.
	The BULK provider in the OPENROWSET command with the SINGLE_CLOB option.
	The BULK provider in the OPENROWSET command with the SINGLE_NCLOB option.
	None of the above.

Module Review and Takeaways

In this module, you have learned how to import and export data, and how to use data-tier applications to simplify the deployment and upgrade of application databases.

When planning a data transfer solution, consider the following best practices:

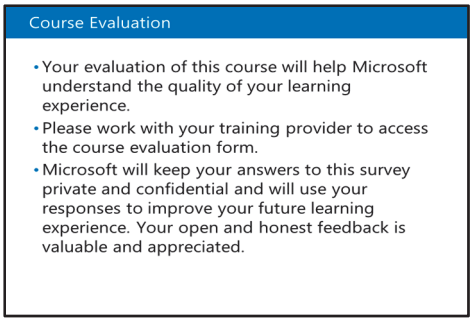
Best Practice:

- Choose the right tool for bulk imports.
- Use SSIS for complex transformations.
- Use **bcp** or BULK INSERT for fast imports and exports.
- Use OPENROWSET when data needs to be filtered before it is inserted.
- Try to achieve minimal logging to speed up data import.

Review Question(s)

Question: What other factors should you consider when importing or exporting data?

Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.