MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official
Curriculum

Module 6: Backing Up Databases

Contents

Microsoft

Development Lead: Xandria Eykel
Technical Lead: Rick Byham
Instructional Designers: Cheryl Hoople, Lin Joyner (Content Master Ltd), Marilyn McGill (Independent Contractor), Gordon Ritchie (Content Master Ltd.),
Subject Matter Experts: Karl Dehmer, Mike Galos, Graeme Malcolm (Content Master), Mary Neville (Content Master Ltd), and Carl Rabeler (Shadow Mountain Computers),
Classroom Automation: Lorrin Smith-Bates
Graphic Artist: Kimberly Jackson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Bill Jones (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Ed Casper (S&T Consulting), Theano Petersen (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: Testing Testing 123
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Database Management: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
90 Minutes

Lab:
45 Minutes

This module provides students with the fundamentals of backing up Microsoft® SQL Server™ 2000 databases, as well as suggestions on when to back up databases and the steps to perform backups. After students familiarize themselves with the different SQL Server backup methods, they will be able to determine a backup strategy that is appropriate for their particular business environments.

In the lab, students will have an opportunity to create backup files that store the backups and to perform full database, differential, and transaction log backups.

After completing this module, students will be able to:

- Create backup files and backup sets.
- Set and change a database recovery model.
- Back up user and system databases by using Transact-SQL and SQL Server Enterprise Manager.
- Back up databases that are created on multiple files and filegroups.
- Use the BACKUP LOG statement to back up and clear transaction logs.
- Apply the appropriate backup options to each of the different SQL Server backup methods.
- Design an appropriate backup strategy.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2073A_06.ppt.
- The C:\Moc\2072A\Demo\D06_Ex.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.
- Practice the presentation, including the animated slide.
- Review any relevant white papers located on the Trainer Materials compact disc.

Demonstration

The following demonstrations are completed in this module:

Using SQL Server Enterprise Manager to Perform Backups

In this demonstration, you will perform a full database backup of the **Northwind** database to a backup file on disk. As you navigate through the interface, map the interface to the Transact-SQL statements.

► To back up a database with SQL Server Enterprise Manager

Use the information in the following table to back up the **Northwind** database.

Option	Value
Database name	**Northwind**
Name of backup	NorthwindFull
Description	A full backup of **Northwind**
Backup type	Full database (complete)
Destination of backup file	Disk
Backup device	C:\ Backup\Demo.bak
Overwrite existing media	Selected
Initialize and label media	Selected
Media set name	DemoBackupFile
Media description	Single file that contains a backup of **Northwind**

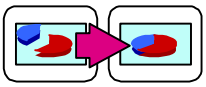
► To map the interface to the Transact-SQL statements and options

Use the information in the following table to point out the Transact-SQL statements and options that map to the elements of the interface.

Interface element	Maps to Transact-SQL
Database—Complete	BACKUP DATABASE statement
Database—Differential	BACKUP DATABASE WITH DIFFERENTIAL statement
Transaction Log	BACKUP LOG statement
File and Filegroup	BACKUP DATABASE FILE statement
Backup to Disk File name	BACKUP DATABASE TO DISK statement (A temporary backup file is created)
Backup to Backup Device	**sp_addumpdevice** system stored procedure (A permanent backup file is created)
Append to media	WITH NOINIT option
Overwrite existing media	WITH INIT option
Eject tape after backup	WITH UNLOAD option
Initialize and label media	WITH FORMAT and NAME options

Other Activities

This section provides procedures for implementing interactive activities to present or review information, such as games or role playing exercises.



Displaying the Animated PowerPoint Slide

All animated slides are identified with an icon of links on the lower left corner of the slide.

► **To display the Using Multiple Backup Files to Store Backups slide**

1. Display the topic slide.

   Database A and Database B appear. The graphic shows that Database A is backed up into three files (File1, File2, and File3).

   Explain that the backups that are striped across the three files make up a backup set. File1, File2, and File3 store parts of an entire backup that make up a media set. File1, File2, and File3 must always be used together.

2. Advance to the next animation to illustrate that backing up Database B to File3 cannot occur, because File1, File2, and File3 must always be used together.
3. Advance to the next animation where the images return to a consistent state.
4. Advance to the next animation to illustrate that backing up Database B across File1, File2, and File3 is acceptable.

   Emphasize that when you use multiple backup files to store backups, you must use the backup files together.

Module Strategy

Use the following strategy to present this module:

- Preventing Data Loss

  Discuss events that can cause data loss or corruption, and describe how an appropriate backup strategy can minimize damage. Also point out that backing up frequently can help ensure data consistency in the event of a system failure.

- Setting and Changing a Database Recovery Model

  Emphasize that it is important to select the appropriate recovery model based on performance requirements, storage requirements, and protection against data loss.

- SQL Server Backup

  Describe the SQL Server dynamic back up process and explain what data is backed up, who can perform backups, and where to store backups.

- When to Back Up Databases

  Point out that both system and user databases can be backed up. Although a business environment determines when to back up databases, this module discusses specific situations when system and user databases should be backed up. Inform students that SQL Server allows backups to occur while users are online; then describe selected activities that interfere with the backup process.

- Performing Backups

  Explain that before students can perform a backup, they first must create the backup files that are necessary to store the backups. SQL Server creates both permanent and temporary backup files. SQL Server also offers the ability to back up to multiple files, so explain the factors that may affect a decision to use this approach. Focus on the BACKUP statement and the typical options that students will use when they perform backups. Then, identify the specific issues that they must consider when they back up to a tape device.

- Types of Backup Methods

  Describe the different types of SQL Server backup methods: full database, differential, transaction log, and database file or filegroup. Discuss the BACKUP LOG statement and the various options that are available to back up, as well as clear, transaction logs. Point out the specific requirement for indexes when performing database file or filegroup backups.

  Demonstrate backing up a database with SQL Server Enterprise Manager and point out the Transact-SQL statement and options that map to the interface.

- Planning a Backup Strategy

  After students are familiar with the fundamentals of backing up databases and the different backup methods, apply the topics that are discussed in the module to the planning of a backup strategy.

  Choosing a backup strategy is dependent on the restore operation, so present the provided scenarios with an emphasis on the backup strategy rather than on the restore operation. Restoring a database is discussed in the next module.

Customization Information

This section identifies the lab setup configuration changes that occur on student computers during the lab. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2072A, *Administering a Microsoft SQL Server 2000 Database.*

Lab Setup

The lab in this module requires that SQL Server 2000 Enterprise Edition has been installed on student computers. To prepare student computers to meet this requirement, perform exercise 1 in lab A, "Installing SQL Server," in module 2, "Planning to Install SQL Server" of course 2072A, *Administering a Microsoft SQL Server 2000 Database*.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
Having a backup strategy is critical to maintaining a database

- **Preventing Data Loss**
- **Setting and Changing a Database Recovery Model**
- **SQL Server Backup**
- **When to Back Up Databases**
- **Performing Backups**
- **Types of Backup Methods**
- **Planning a Backup Strategy**

This module provides the fundamentals of backing up Microsoft® SQL Server™ 2000 databases, as well as suggestions on when to back up databases and the steps to perform backups. After you learn the different SQL Server backup methods, you will be able to determine a backup strategy that is appropriate for your particular business environment.

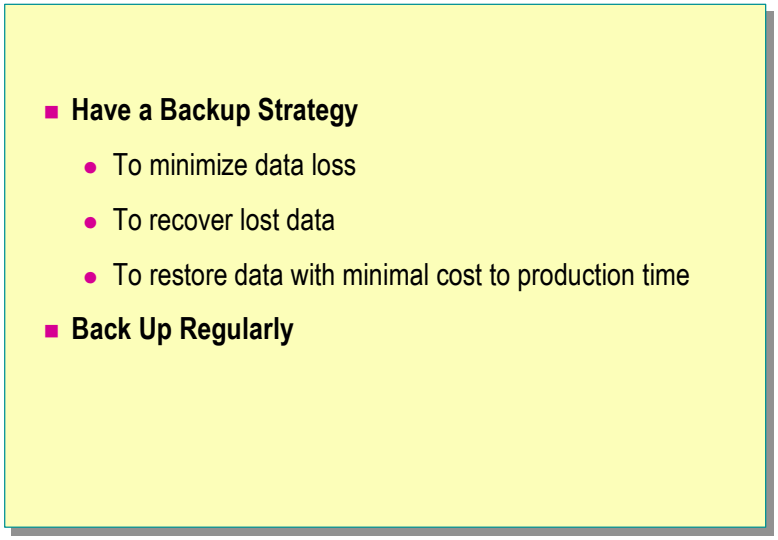
After completing this module, you will be able to:

- Create backup files and backup sets.
- Set and change a database recovery model.
- Back up user and system databases by using Transact-SQL and SQL Server Enterprise Manager.
- Back up databases that are created on multiple files and filegroups.
- Use the BACKUP LOG statement to back up and clear transaction logs.
- Apply the appropriate backup options to each of the different SQL Server backup methods.
- Design an appropriate backup strategy.

Preventing Data Loss

Topic Objective
To discuss the importance of preventing data loss.

Lead-in
Preventing data loss is one of the most critical issues that you face as system administrators.



Preventing data loss is one of the most critical issues that system administrators encounter.

Have a Backup Strategy

You must have a backup strategy to minimize data loss and recover lost data. You can lose data as a result of hardware or software failures or:

- Accidental or malicious use of the DELETE statement.
- Accidental or malicious use of the UPDATE statement—for example, not using a WHERE clause with the UPDATE statement (all rows are updated rather than a single row in a particular table).
- Destructive viruses.
- Natural disasters, such as fire, flood, and earthquakes.
- Theft.

If you have an appropriate backup strategy, you can restore data with minimal cost to production time and minimize the chance of permanent data loss. Think of a backup strategy as an insurance policy. Your backup strategy should put your system back to where it was before a problem occurred. As with an insurance policy, ask yourself, “How much am I willing to pay, and how much loss is acceptable to me?”

The costs that are associated with a backup strategy include the amount of time that is spent designing, implementing, automating, and testing the backup procedure. Although you cannot prevent data loss completely, you should design your backup strategy to minimize the extent of the damage. When you plan your backup strategy, consider the acceptable amount of time that the system can be down, as well as the acceptable amount of data loss (if any) in the event of a system failure.

Back Up Regularly

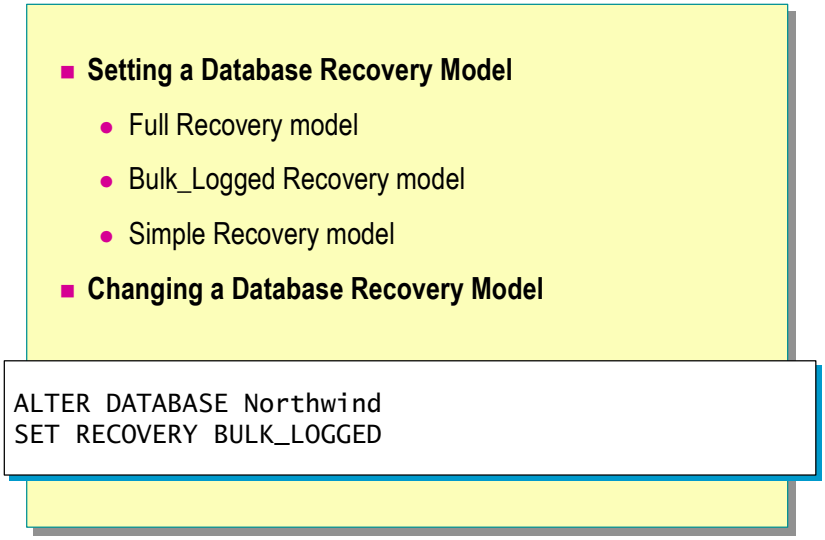
How frequently you back up your database depends on the amount of data that you are willing to lose and the volume of database activity. When you back up user databases, consider the following facts and guidelines:

- You might back up your database frequently if your system is in an online transaction processing (OLTP) environment.
- You might back up your database less frequently if your system has little activity or is used primarily for decision support.
- You should schedule backups when SQL Server is not in the process of being heavily updated.
- After you determine your backup strategy, you can automate the process by using the Database Maintenance Plan Wizard.

Setting and Changing a Database Recovery Model

Topic Objective
To explain the database recovery models.

Lead-in
SQL Server 2000 has three database recovery models.



You can set or change your recovery model at any time, but you should plan a recovery model when you create a database.

Setting a Database Recovery Model

SQL Server 2000 has three database recovery models. Each of the models maintains data in the event of a server failure, but there are key differences in how SQL Server recovers data and in the storage and performance needs in the event of disk failure.

Full Recovery Model

The Full Recovery model is the default.

You can use the Full Recovery model when full recovery from damaged media is the highest priority. This model uses copies of the database and all log information to restore the database. SQL Server logs all changes to the database, including bulk operations and index creations. Provided that the logs themselves are not damaged, SQL Server can recover all data except transactions actually in process at the time of the failure.

Because all transactions are logged, recovery can be made to any point in time. SQL Server 2000 supports the insertion of named marks into the transaction log to allow recovery to that specific mark.

Because log transaction marks consume log space, you should only use them for transactions that play a significant role in the database recovery strategy. The main limitation of this model is the large size of the log files and the resulting storage and performance costs.

Bulk_Logged Recovery Model

Similar to the full recovery model, the Bulk_Logged Recovery model uses both database and log backups to recreate a database. However, the Bulk_Logged Recovery model uses less log space for the following operations: CREATE INDEX, bulk load operations, SELECT INTO, WRITETEXT, and UPDATETEXT. The log notes only the occurrence of these operations as bits in extents instead of storing details of the operations in the log.

To preserve the changes for an entire bulk load operation, extents that are marked as changed are also stored in the log. As a result of only storing the final result of multiple operations, the log is typically smaller and bulk operations can run faster.

Using this model can restore all data, but a disadvantage is that it is not possible to restore only part of a backup, such as restoring to a specific mark.

Simple Recovery Model

You typically use the Simple Recovery model for small databases or databases where data changes infrequently. This model uses full or differential copies of the database and recovery is limited to restoring the database to the point when the last backup was made. All changes made after the backup are lost and need to be recreated. The principal benefit of this model is that it takes less storage space for logs and is the simplest model to implement.

Changing a Database Recovery Model

By default, SQL Server 2000 Standard Edition and SQL Server 2000 Enterprise Edition use the Full Recovery model. You can change the recovery model at any time, but you must make an additional backup at the time of the change. To find out which model your database is using, use the DATABASEPROPERTYEX function.

Syntax

ALTER DATABASE *database_name*
  SET RECOVERY {FULL | SIMPLE | BULK_LOGGED}

Example

This example sets the recovery of the **Northwind** database as the Bulk_Logged Recovery model.

```
ALTER DATABASE Northwind SET RECOVERY BULK_LOGGED
```

SQL Server Backup

Topic Objective
To introduce SQL Server backup.

Lead-in
SQL Server backup is dynamic.

- **Allows Backups to Occur While Users Continue to Work with the Database**
- **Backs Up Original Files and Records Their Locations**
- **Captures Database Activities That Occur During the Backup Process in the Backup**
  - Issues a checkpoint and records the LSN
  - Writes all pages to the backup media
  - Writes all transaction log records written during the backup process

Key Point
Point out that database activities that occur during the backup process are included in the backup.

During the backup operation, SQL Server:

- Allows you to perform database backups while users continue to work with the database.
- Backs up the original database files and records their locations. The backup contains:
- Schema and file structure.
- Data.
- Portions of the transaction log files. The portion of the transaction log that is backed up contains database activities since the start of the backup process.

  SQL Server uses these backups to recreate the files in their original locations, complete with objects and data, when you restore a database.

Key Point
Explain that the SQL Server backup process is dynamic and describe how this is accomplished.

- Captures database activities that occur during the backup process.

  The SQL Server backup process is dynamic and, with some exceptions, can occur while the database is online and being actively modified. The dynamic backup process is accomplished when SQL Server:

- Issues a checkpoint on the database and records the log sequence number (LSN) of the oldest active transaction log record.
- Writes all pages to the backup media by reading the disks directly (bypassing the buffer cache).
- Writes all transaction log records written during the backup process. Specifically, SQL Server writes transaction log records from the recorded LSN through the end of the log.

Performing and Storing Backups

Topic Objective
To provide a basic overview of SQL Server backup.

Lead-in
To back up a database in SQL Server, you must consider who is allowed to perform the backup and where to store it.

- **Who Performs Backups**
  - Members of the **sysadmin** fixed server role
  - Members of the **db_owner** and **db_backupoperator** fixed database roles
- **Where to Store Backups**
  - Hard disk file
  - Tape
  - A location identified by a Named Pipe

To back up a database in SQL Server, you must consider who is allowed to perform the backup and where to store it. You can back up databases by executing Transact-SQL statements or by using SQL Server Enterprise Manager.

Who Performs Backups

Members of the following roles have permission to back up a database:

- The **sysadmin** fixed server role
- The **db_owner** fixed database role
- The **db_backupoperator** fixed database role

Additional roles can be created and granted permission to back up a database.

Where to Store Backups

SQL Server can back up to a hard disk file, tape, or Named Pipe.

- Disk files (local or network) are the most common media that is used for storing backups.
- When you back up to a tape, the tape drive must be attached locally to SQL Server.
- SQL Server provides the ability to back up to a Named Pipe in order to allow users to take advantage of the back up and restore features of third-party software packages.

When to Back Up Databases

Topic Objective
To provide an overview of deciding when to back up databases.

Lead-in
We'll discuss specific situations in which you should back up system and user databases, as well as activities that interfere with backing up databases.

- Backing Up System Databases
- Backing Up User Databases
- Activities That Are Restricted During Backup

When and how often you back up your database depends on your particular business environment. However, there are times when you may need to supplement your backup strategy. For instance, you may occasionally need to back up the system databases or a specific user database.

Although SQL Server backup is dynamic, some activities cannot occur on the database during backup operations.

Backing Up System Databases

Topic Objective
To explain when to back up the system databases.

Lead-in
You should back up your system databases regularly.

- **After Modifying the master Database**
  - Using the CREATE DATABASE, ALTER DATABASE, or DROP DATABASE statement
  - Executing certain system stored procedures
- **After Modifying the msdb Database**
- **After Modifying the model Database**

System databases store important data about SQL Server and all user databases. Therefore, you should back up system databases regularly, as a matter of course, and specifically, whenever you modify them.

After Modifying the master Database

The **master** database contains information about all databases on a SQL Server. Back up the **master** database when *any* user-defined databases are created. This enables you to recover more easily and to restore user databases if the **master** database becomes damaged.

After the **master** database is rebuilt and restored, you can restore other system database backups and reference existing user databases.

Note Without a current backup of the **master** database that contains references to user databases, you must completely rebuild all of the system databases with the **rebuildm** command-prompt utility by running 80\Tools\Binn\Rebuildm.exe. This utility rebuilds all system databases as a unit.

When you execute certain statements or system stored procedures, SQL Server modifies the **master** database automatically. Therefore, back up the **master** database when you execute the following:

- The CREATE DATABASE, ALTER DATABASE, or DROP DATABASE statement that creates, alters, or removes a database
- The **sp_logdevice** system stored procedure, which alters the transaction log
- The **sp_addserver**, **sp_dropserver**, and **sp_addlinkedserver** system stored procedures, which add or drop servers
- The **sp_addmessage** system stored procedure, or adding error messages with the SQL Server Enterprise Manager.

Key Point
Briefly discuss the system stored procedures and explain why backing up the **master** database is recommended.

After Modifying the msdb Database

Back up the **msdb** database after you modify it, because **msdb** contains information about jobs, alerts, and operators that are used by SQL Server Agent. If you do not have a current backup of the **msdb** database, you must rebuild all of the system databases in the event of a system failure and then recreate each job, alert, and operator.

After Modifying the model Database

Back up the **model** database if you modify it to include the default configuration for all new user databases. Because user databases are rebuilt when the **master** or **msdb** databases are rebuilt, changes to the **model** database are also lost. You can restore a backup of your customized **model** database in the event of a system failure.

Backing Up User Databases

Topic Objective
To describe when user databases should be backed up.

Lead-in
Similar to system databases, you should back up all user databases on a regular basis, and specifically, after certain operations.

- **After Creating Databases**
- **After Creating Indexes**
- **After Clearing the Transaction Log**
- **After Performing Nonlogged Operations**
  - BACKUP LOG WITH TRUNCATE_ONLY or NO_LOG statement
  - WRITETEXT or UPDATETEXT statement
  - SELECT...INTO statement

You should plan to back up user databases regularly. You also should perform a backup after a database or index is created and when certain nonlogged operations are executed.

After Creating Databases

You should back up a database after it has been created or loaded with data. Without a full database backup, you cannot restore transaction log backups because you must have a baseline to which the transaction logs can be applied.

After Creating Indexes

You should back up the database whenever you create an index. Although you are not required to do so, if the database is lost, you will save time during the restore process. Backing up a database after an index is created ensures that the database backup file contains the data and the index structures.

If you back up only the transaction log after an index is created, and you then restore that transaction log at some point in the future, SQL Server must rebuild the index. The amount of time that is required to rebuild the index may be longer than the time it takes to restore a full database backup.

Note Be aware that the transaction log only records the fact that an index was created, not the actual data page modifications.

After Clearing the Transaction Log

You should back up a database after you clear the transaction log with the BACKUP LOG WITH TRUNCATE_ONLY or the BACKUP LOG WITH NO LOG statement. When you execute these statements, the transaction log no longer contains a record of database activity and cannot be used to recover changes to the database.

After Performing Nonlogged Operations

Operations that are not recorded to the transaction log are called nonlogged operations.

With some recovery models you cannot recover changes made by the following nonlogged operations:

- BACKUP LOG WITH TRUNCATE ONLY or BACKUP LOG WITH NO_LOG statement. SQL Server removes the inactive part of the transaction log without making a backup copy. Additionally, the act of truncating the transaction log is not recorded in the transaction log.
- WRITETEXT or UPDATETEXT statement. SQL Server modifies data in text columns and, by default, does not record this activity in the transaction log. However, you can specify the WITH LOG option to write these activities to the transaction log.
- SELECT…INTO statement when creating a permanent table or the bulk copy program.

Back up a database after you perform a nonlogged operation, because if your system fails, the transaction log might not contain the information needed to restore the database to a consistent state.

Activities That Are Restricted During Backup

Topic Objective
To identify some activities that interfere with database backups.

Lead-in
You can back up a database while the database is online and active. However, some operations cannot take place during a backup.

- **Creating or Modifying Databases**
- **Performing Autogrow Operations**
- **Creating Indexes**
- **Performing Nonlogged Operations**
- **Shrinking a Database**

You can back up a database while the database is online and active. However, a few operations are not advisable during the backup process.

Avoid the following activities during database backups:

- Creating or modifying databases with the CREATE DATABASE or ALTER DATABASE statement.
- Performing autogrow operations.
- Creating indexes.
- Performing any nonlogged operations, including a bulk load of data and the SELECT…INTO, WRITETEXT, and UPDATETEXT statements.
- Shrinking a database.

Performing Backups

Topic Objective
To introduce the steps that are involved in performing a backup.

Lead-in
Now we will discuss the specific steps in performing a backup.

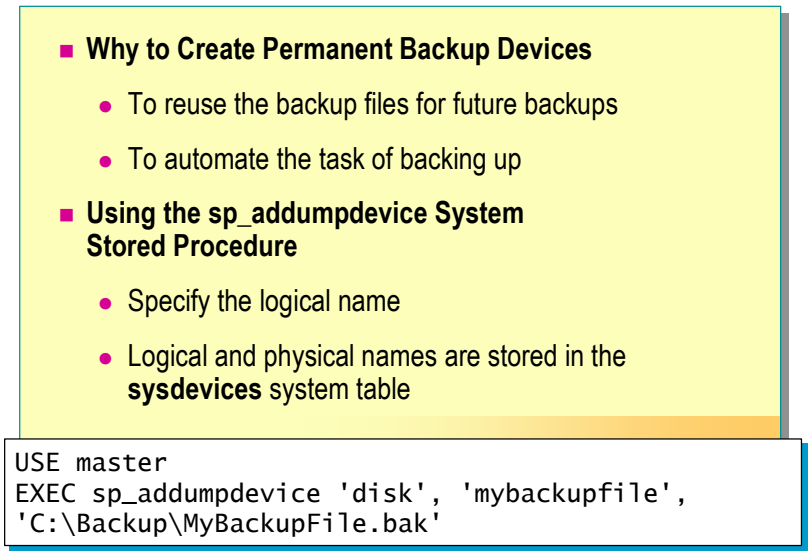
- Creating Backup Devices
- Creating Backup Files Without Permanent Devices
- Using Multiple Backup Files to Store Backups
- Using the BACKUP Statement
- Backing Up to a Tape Device

When you perform a backup, you first must create the backup files (permanent or temporary) to contain your backup. SQL Server provides options that you can apply to each of the different backup methods that are available to you. Although SQL Server also allows you to choose a number of backup destinations, backing up to a disk or tape is the most common.

Creating Backup Devices

Topic Objective
To discuss creating backup devices.

Lead-in
The first step in performing a backup is to create the backup devices that will contain your backup.



The first step in performing a backup is to create the backup files that will contain your backup. A backup file that is created before it is used for a backup operation is called a backup device.

Delivery Tip
Use SQL Server Enterprise Manager to demonstrate how to create backup devices.

A device does not create a backup file; the file is created on first use of the device.

Why to Create Permanent Backup Devices

If you want to reuse the backup files that you create or to automate the task of backing up your database, you must create permanent backup devices. You can create backup devices with SQL Server Enterprise Manager or by executing the **sp_addumpdevice** system stored procedure.

Using the sp_addumpdevice System Stored Procedure

Execute the **sp_addumpdevice** system stored procedure to create backup devices on a disk or tape or to direct data to a Named Pipe. When you create backup devices, consider the following facts:

- SQL Server creates logical and physical names in the **sysdevices** system table of the **master** database.
- You must specify the logical and physical names of the backup file.
- You can create up to 64 backup files for a database.

When you create a new backup device with SQL Server Enterprise Manager, SQL Server executes the **sp_addumpdevice** system stored procedure for you.

Syntax

sp_addumpdevice [**@devtype** =] '*device_type*',
[**@logicalname** =] '*logical_name*',
[@physicalname =] '*physical_name*'
[,{ [**@cntrltype** =] *controller_type* | [**@devstatus** =] '*device_status*'}]

Where device_type is {DISK | TAPE | PIPE}

Example 1

This example creates a permanent backup file on a hard disk.

```
USE master
EXEC sp_addumpdevice 'disk', 'mybackupfile',
'C:\ Backup\MyBackupFile.bak'
```

Example 2

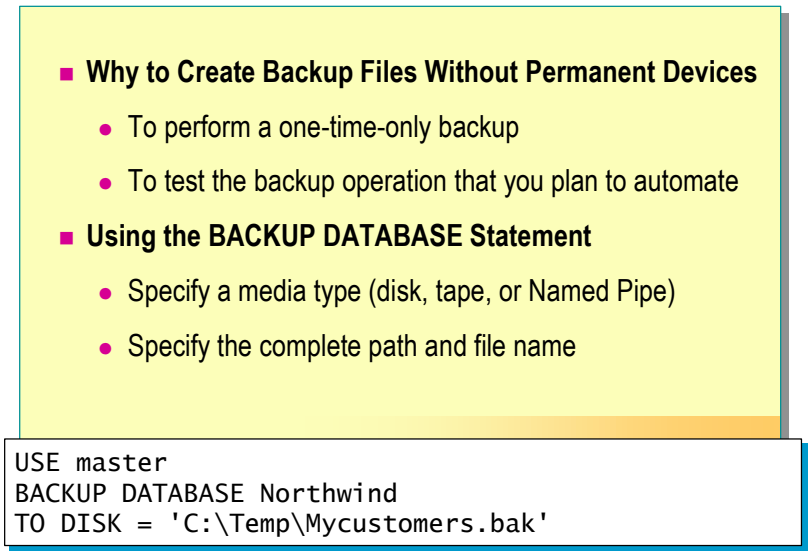
This example creates a backup device on a tape with the logical name **Mytape1** and the physical name \\.**tape0**

```
USE master
EXEC sp_addumpdevice 'tape', 'mytape1', '\\.\tape0'
```

Creating Backup Files Without Permanent Devices

Topic Objective
To discuss creating temporary backup files by using the BACKUP DATABASE statement.

Lead-in
While creating a permanent backup device is preferable, you also can create temporary backup files.



While creating a permanent backup device is preferable, you also can create temporary backup files with the BACKUP DATABASE statement without having to specify a backup device.

Why to Create Backup Files Without Permanent Devices

If you do not plan to reuse the backup files, create a backup file without a permanent device. For example, if you are performing a one-time-only backup of a database or are testing the backup operation that you plan to automate, you may want to create a temporary backup file.

Using the BACKUP DATABASE Statement

Create temporary backup files with the BACKUP DATABASE statement or with SQL Server Enterprise Manager. Before SQL Server performs a backup, it creates a backup file to store the results of a backup operation. The temporary backup file must not exist prior to performing the backup.

If you create a temporary backup file, you must:

- Specify a media type (disk, tape, or Named Pipe).
- Specify the complete path and file name.

Partial Syntax

BACKUP DATABASE {*database_name* | **@***database_name_var*}
TO <backup_device> [*, ...n*]

Where <backup_device> is:
{{*backup_device_name* | **@***backup_device_name_var*} | {DISK | TAPE | PIPE} =
{'*temp_backup_device*' | **@***temp_backup_device_var*}

Example

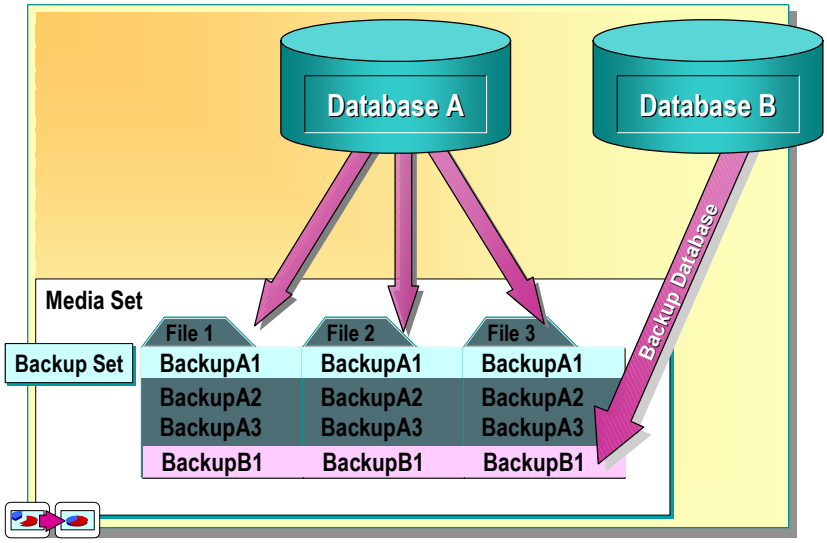
This example creates a temporary backup file on a disk and backs up the **master** database onto the temporary backup file.

```
USE master
BACKUP DATABASE Northwind TO DISK = 'C:\Temp\MyCustomers.bak'
```

Using Multiple Backup Files to Store Backups

Topic Objective
To discuss using multiple backup files to store backups.

Lead-in
SQL Server can write to multiple backup files at the same time (in parallel). You can back up to a backup set to reduce the total time that it takes to back up and restore your database.



**********ILLEGAL FOR NON-TRAINER USE**********

Delivery Tip
This slide is animated. Refer to the Instructor Notes if you require help in navigating through this slide.

SQL Server can write to multiple backup files at the same time (in parallel). When you have multiple backup files, data is striped across all files that are used to create the backup. These files store a striped backup set. A backup set is a result of a single backup operation on single or multiple files.

Storing Backups on Multiple Backup Files

Back up to multiple tapes or disk controllers to decrease the total time that it takes to back up a database. For example, if a backup operation that uses one tape drive normally takes four hours to complete, you can add a second tape drive and reduce the length of the backup operation to only two hours.

Partial Syntax

```
BACKUP DATABASE {database_name | @database_name_var}
  TO <backup_device> [, ...n]
  [WITH
    [MEDIANAME = {media_name | @medianame_var}]
  ]
```

Key Point
Point out that when you use multiple files to back up a database, you should use the MEDIANAME option to associate these striped sections of a backup with one another as members of a backup set.

When you use multiple files to store your backups, consider the following facts:

- All devices that are used in a single backup operation must be of the same media type (disk or tape). You cannot mix disk and tape devices for a single backup media set. A media set is a collection of files that are used to contain one or more backup sets.
- You can use a combination of permanent and temporary files when you create a backup set.
- If you define a file as a member of a backup set, you must always use the files together.

- You cannot use only one member of the backup set for a backup operation unless you reformat the files.
- If you reformat one member of a backup set, the data that is contained in the other members of the backup set is invalid and unusable.

For example, if a striped backup set was created on two files, all subsequent backup operations that involve the same backup set must use these same two files as well. You can append additional backups to these two files. However, if you want to use only one of these files to back up another database or to use as part of another backup set, you must reformat the file.

Note If you are using multiple devices, each backup file has a family designation, such as Family 1, that identifies the device that created the file.

Using the MEDIANAME Option

The MEDIANAME option specifies the name for the entire backup media set. When you use multiple files to back up a database, you should use the MEDIANAME option. The MEDIANAME option associates the multiple files with one another as members of a media set.

After the media set has been created and named, you can reuse the media set for future backup operations. Names may have up to 128 characters.

Using the BACKUP Statement

Topic Objective
To introduce the BACKUP statement and Transact-SQL options that are used for all backup methods.

Lead-in
SQL Server Enterprise Manager makes it easy for you to perform backups.

- **Specifying the INIT or NOINIT Option**
  - NOINIT option appends to a backup file
  - INIT option overwrites a backup file
- **Using the FORMAT Option**
  - Overwrites the contents of a backup file
  - Splits up a striped backup set

**********ILLEGAL FOR NON-TRAINER USE**********

You can perform backup operations with SQL Server Enterprise Manager, the Backup Wizard, or Transact-SQL. You should be familiar with the backup options that are available when you use any of the SQL Server backup methods.

Syntax

```
BACKUP DATABASE {database_name | @database_name_var}
  TO <backup_device> [, ...n]
  [WITH
    [FORMAT]
    [[,] {INIT | NOINIT}]
  ]
```

Specifying the INIT or NOINIT Option

When you back up a database, determine whether to overwrite or append to a backup file:

- The SQL Server default is to append (NOINIT) backups to a file. If you use the NOINIT option, SQL Server appends a backup to an existing backup file or backup set.
- If you use the INIT option, SQL Server overwrites any existing data on the backup media set but retains the header information. If the first file of the backup set on the device has an ANSI-standard label, SQL Server determines whether the previous backup set can be overwritten.

The backup operation fails and data is not overwritten if:

- The EXPIREDATE option that you specified on the backup device has not yet expired.
- The *backup_set_name* parameters that you specified in the NAME option do not match the *backup_set_name* in the backup device.
- You attempt to overwrite one member of a previously named backup set. SQL Server detects that the file is a member of a backup set.

Using the FORMAT Option

Use the FORMAT option to overwrite the contents of a backup file and split up the backup set:

- A new media header is written on all files that are used for this backup operation.
- SQL Server overwrites both the existing media and the contents of the backup file.
- Use the FORMAT option carefully. Formatting only one backup file of a media set renders the entire backup set unusable.

  For example, if a single tape that contains a part of an existing striped backup set is reformatted, the entire backup set is unusable.

Backing Up to a Tape Device

Topic Objective
To discuss details that are involved with backing up to a tape device.

Lead-in
You should be aware of specific characteristics when you back up to a tape device.

- Requires Tape to Be Attached Locally to SQL Server
- Records Backup Information on Tape Label
- Stores SQL Server and Non-SQL Server Backups

Tapes are a convenient medium for backups because they are inexpensive, provide a large amount of storage, and can be stored off-site for data safety and security.

Requires Tape to Be Attached Locally to SQL Server

When you back up to a tape, the tape drive must be attached locally to SQL Server.

Records Backup Information on Tape Label

When you back up to a tape, SQL Server records backup information on the tape label, which includes the:

- Database name
- Time
- Date
- Type of backup

Stores SQL Server and Non-SQL Server Backups

SQL Server uses a standard backup format called Microsoft Tape Format to write backups to tapes. As a result, both SQL Server and non-SQL Server data can be backed up to the same tape.

SQL Server backups can coexist on the same media as other backup sets or as backup sets that are produced by other clients that use this standard format. For example, both SQL Server backups and Microsoft Windows® 2000 backups can exist on the same tape.

Specifying Tape Options

Topic Objective
To discuss the specific options for using tapes to store backups.

Lead-in
When you back up to a tape, you can use options that are specific to this type of backup medium.

Tape option	Description
UNLOAD (default)	Rewinds and unloads the tape
NOUNLOAD	Does not rewind and unload the tape
BLOCKSIZE	Changes the physical block size in bytes
FORMAT	Writes a header on files that are used for a backup
SKIP	Ignores ANSI tape labels
NOSKIP (default)	Reads ANSI tape labels
RESTART	Restarts the backup operation from the point of interruption

When you back up to a tape, you can use options that are specific to this type of backup medium.

UNLOAD

SQL Server automatically rewinds and unloads the tape from the tape drive after the backup is complete. The UNLOAD option is the SQL Server default and remains set until you select the NOUNLOAD option.

NOUNLOAD

Use this option if you do not want SQL Server to rewind and unload the tape medium from the tape drive automatically after a backup. The NOUNLOAD option remains set until you select UNLOAD.

BLOCKSIZE

Use this option to change the physical block size in bytes if you are overwriting a tape medium with the FORMAT or SKIP and INIT options. When you back up to a tape, SQL Server selects an appropriate block size. You can override the block size selection by using the BLOCKSIZE option and specifying a block size.

FORMAT

Use this option to write a header on all of the volumes (files) that are used for a backup. SQL Server overwrites all headers and backups on the files. The header includes information that is found in the MEDIANAME and MEDIADESCRIPTION options.

When you use the FORMAT option to back up to a tape device, the INIT and SKIP options are implied, and, therefore, you do not need to specify these options.

SKIP

Use this option to skip headers. SQL Server ignores any existing ANSI tape labels on the tape device. The ANSI label of a tape can provide warning information about the expiration date of the tape, as well as enforce write permissions.

NOSKIP

Use this option if you want SQL Server to read ANSI tape labels. SQL Server will check the expiration date and name of all backup sets on the media before allowing them to be overwritten. SQL Server reads ANSI tape labels by default.

RESTART

Use this option to restart the backup operation from the point of interruption for tape backups that span multiple tape volumes. You must restart the backup process manually by executing the original BACKUP statement with the RESTART option.

Types of Backup Methods

Topic Objective
To provide an overview of the different SQL Server backup methods.

Lead-in
SQL Server provides you with different backup methods.

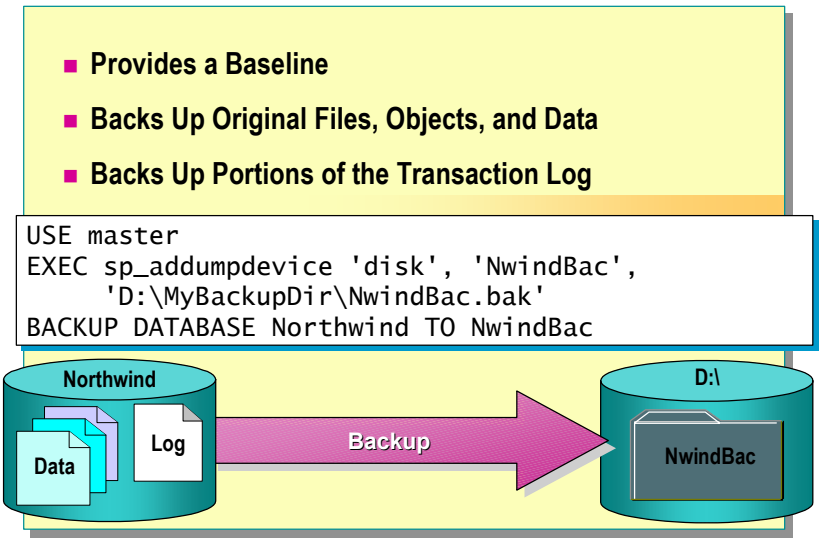
- Performing a Full Database Backup
- Performing a Differential Backup
- Performing a Transaction Log Backup
- Performing a Database File or Filegroup Backup

SQL Server provides different backup methods to meet the needs of a wide range of business environments and database activities.

Performing a Full Database Backup

Topic Objective
To discuss how to perform a full database backup and how SQL Server processes this type of backup.

Lead-in
You should perform a full database backup periodically, because you must have a baseline from which you can recover in the event of system failure.



**********ILLEGAL FOR NON-TRAINER USE**********

Delivery Tip
Use SQL Server Enterprise Manager to demonstrate how to perform a full database backup.

If your database is primarily a read-only database, full database backups may be sufficient to prevent data loss. A full database backup serves as your baseline in the event of a system failure. When you perform a full database backup, SQL Server:

- Backs up any activity that took place during the backup.
- Backs up any uncommitted transactions in the transaction log.

  SQL Server uses the portions of the transaction log that were captured in the backup file to ensure data consistency when the backup is restored.

Example 1

This example creates a named backup device with the logical name Nwindbac and performs a full database backup.

```
USE master
EXEC sp_addumpdevice 'disk', 'NwindBac',
     'D:\MyBackupDir\NwindBac.bak'
BACKUP DATABASE Northwind TO NwindBac
```

Example 2

This example performs a full database backup to the NwindBac file and overwrites any previous backups on that file.

```
BACKUP DATABASE Northwind TO NwindBac WITH INIT
```

Example 3

This example appends a full database backup to the NwindBac file. Any previous backup files are left intact.

```
BACKUP DATABASE Northwind TO NwindBac WITH NOINIT
```

Example 4

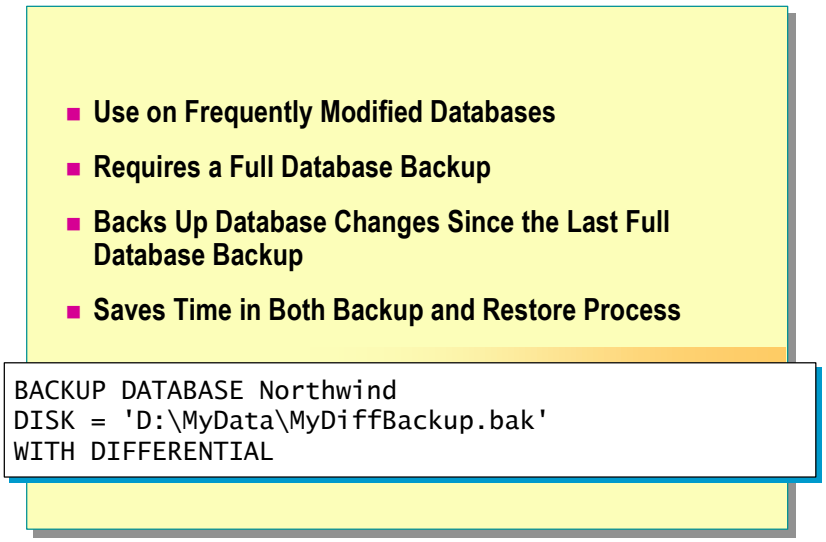
This example creates a backup disk file and performs a full database backup to that file.

```
BACKUP DATABASE Northwind TO
DISK = 'D:\Temp\MyTempBackup.bak'
```

Performing a Differential Backup

Topic Objective
To discuss how to perform a differential backup and how SQL Server processes this type of backup method.

Lead-in
You should perform a differential backup on a database when data is heavily modified; a differential backup offers the advantages of a smaller backup set and faster restore time.



You should perform a differential backup to minimize the time that is necessary for restoring a frequently modified database. Perform a differential backup *only* if you have performed a full database backup. In a differential backup, SQL Server:

- Backs up the parts of the database that have changed since the last full database backup.

  To determine which pages have changed since the last full database backup, SQL Server compares the LSN on a page to the synchronization LSN of the last full database backup.

  When performing a differential backup, SQL Server backs up extents rather than individual pages. An extent is backed up when the LSN on any page in the extent is greater than the LSN of the last full database backup.
- Backs up any activity that took place during the differential backup, as well as any uncommitted transactions in the transaction log.

Key Points
Point out that a naming convention should be used for backup files that contain differential backups to distinguish them from files that contain full database backups.

When you perform a differential backup, consider the following facts and guidelines:

- If a certain row in the database has been modified several times since the last full database backup, the differential backup contains only the last set of values for that row. This is different from a transaction log backup that contains a history of all changes to that row.
- You minimize the time that is required to back up a database because the backup sets are smaller than they are in full backups.
- You minimize the time that is required to restore a database because you do not have to apply a series of transaction logs.
- You should establish a naming convention for backup files that contain differential backups to distinguish them from files that contain full database backups.

Partial Syntax

BACKUP DATABASE {*database_name* | **@***database_name_var*}
  TO <*backup_device*> [**,** *...n*]
  [WITH
    [DIFFERENTIAL]
  ]

Example

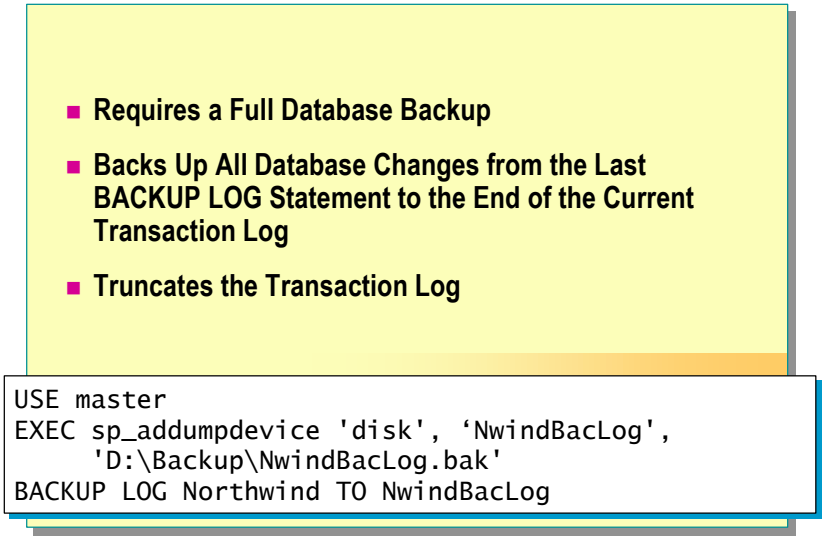
This example creates a differential backup on a temporary backup file.

```
BACKUP DATABASE Northwind TO
DISK = 'D:\MyData\MyDiffBackup.bak'
WITH DIFFERENTIAL
```

Performing a Transaction Log Backup

Topic Objective
To discuss how to perform a transaction log backup and how SQL Server processes this type of backup.

Lead-in
You perform transaction log backups to record any database changes. You typically perform transaction log backups when you perform full database backups.



You back up transaction logs to record any database changes. You typically back up transaction logs when you perform full database backups:

- You should not back up a transaction log unless you have performed a full database backup at least once.
- Transaction logs cannot be restored without a corresponding database backup.
- You cannot back up transaction logs when using the Simple Recovery model.

How SQL Server Backs Up the Transaction Log

When you back up the transaction log, SQL Server:

- Backs up the transaction log from the last successfully executed BACKUP LOG statement to the end of the current transaction log.
- Truncates the transaction log up to the beginning of the active portion of the transaction log and discards the information in the inactive portion.

  The active portion of the transaction log starts at the point of the oldest open transaction and continues to the end of the transaction log.

Partial Syntax

```
BACKUP LOG {database_name | @database_name_var}
  TO <backup_device> [, …n]
  [WITH
    [{INIT | NOINIT}]
  ]
```

Example

This example creates a backup device for the log and backs up the transaction log of the **Northwind** database.

```
USE master
EXEC sp_addumpdevice 'disk', 'NwindBacLog',
    'D:\Backup\NwindBacLog.bak'
BACKUP LOG Northwind TO NwindBacLog
```

Using the NO_TRUNCATE Option

Topic Objective
To discuss the NO_TRUNCATE option.

Lead-in
If the database files are damaged or lost, you should back up transaction logs with the NO_TRUNCATE option.

SQL Server:

- **Saves the Entire Transaction Log Even if the Database Is Inaccessible**
- **Does Not Purge the Transaction Log of Committed Transactions**
- **Allows Data to Be Recovered Up to the Time When the System Failed**

Delivery Tip
Point out how to clear inactive entries from the transaction log by using SQL Server Enterprise Manager.

If the database files are damaged or lost, you should back up transaction logs with the NO_TRUNCATE option. Using this option backs up all recent database activity.

SQL Server:

- Saves the entire transaction log (everything that has happened since the last BACKUP LOG statement), even if the database is inaccessible.
- Does not purge the transaction log of committed transactions.
- Allows you to recover data up to the time when the system failed.

When you restore the database, you can restore the database backup and apply the transaction log backup that is created with the NO_TRUNCATE option to recover data.

Clearing the Transaction Log

Topic Objective
To discuss clearing transaction logs.

Lead-in
The BACKUP LOG statement has a dual purpose. In addition to backing up transaction logs, you can use options that clear the transaction log if it becomes full.

- **Use the BACKUP Statement to Clear the Transaction Log**
- **Using the TRUNCATE_ONLY or NO_LOG Option**
  - Cannot recover changes
  - Is not recorded
- **Setting the trunc. log on chkpt. Option**
  - Writes all committed transactions
  - Occurs automatically when set to true

You can use the BACKUP LOG statement with the TRUNCATE_ONLY or NO_LOG option to clear transaction logs. You should back up the transaction log regularly to keep it at a reasonable size:

- If the transaction log is full, users cannot update the databases and you cannot fully restore the database in the event of a system failure. You must clear the transaction log either by performing a full database backup and saving the data, or by truncating the transaction log.
- If backing up the transaction log does not truncate the majority of your transaction log, you may have an old transaction that is open in the transaction log.

Using the TRUNCATE_ONLY or NO_LOG Option

The TRUNCATE_ONLY and NO_LOG options perform the same function. If you want to clear the transaction log and do not want to keep a backup copy of the data, use these options. SQL Server removes the inactive part of the log without making a backup copy of it. The active portion of the transaction log consisting of uncommitted transactions is never truncated.

Consider the following facts and guidelines when truncating a transaction log:

- Clearing the transaction log before you back up the database results in a smaller backup of the full database.
- You cannot recover the changes that were recorded in the transaction log. You should execute the BACKUP DATABASE statement immediately.
- The action of truncating the transaction log is not recorded.

Partial Syntax

BACKUP LOG {*database_name* | **@***database_name_var*}
[WITH {TRUNCATE_ONLY | NO_LOG }]

Example 1

This example uses the BACKUP LOG statement to remove the inactive portion of a transaction log without making a backup copy.

```
BACKUP LOG Northwind WITH TRUNCATE_ONLY
```

Example 2

This example uses the BACKUP LOG statement to remove the inactive portion of a full transaction log without making a backup copy of it.

```
BACKUP LOG Northwind WITH NO_LOG
```

Setting the trunc. log on chkpt. Option

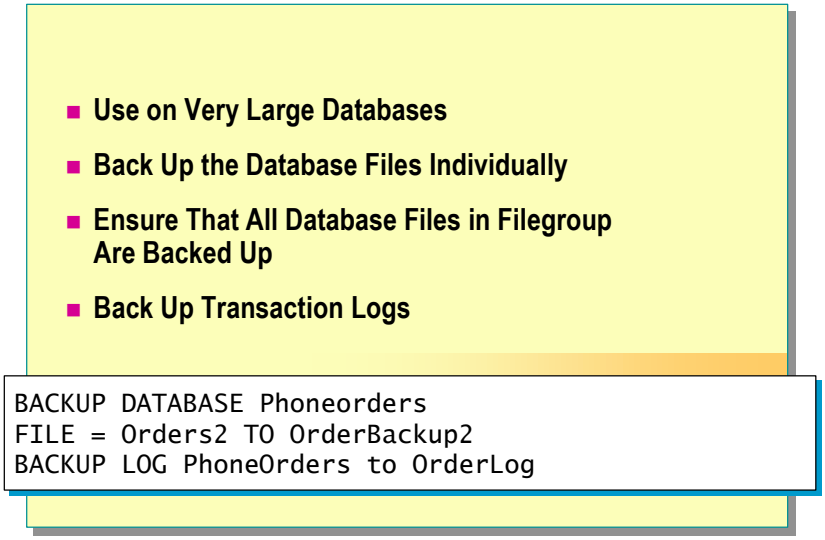
You can set the **trunc. log on chkpt.** option to true to write all committed transactions to the database when a checkpoint occurs. This option automatically truncates the transaction log. The **trunc. log on chkpt.** option is provided for backward compatibility only. The Simple Recovery model replaces it.

Caution If you set the **trunc. log on chkpt.** option to true, you *cannot* back up the transaction log and use it to help restore the database in the event of a system failure. The transaction log no longer stores the changes that are made to the database since the last full database backup.

◆ Performing a Database File or Filegroup Backup

Topic Objective
To discuss how to perform database file backups and how SQL Server processes this type of backup method.

Lead-in
Perform database file or filegroup backups when you have very large databases or when the database must allow updates 24 hours a day. You can back up the critical database files more often.



If performing a full database backup on very large databases (VLDBs) is not practical, you can perform database file or filegroup backups. When SQL Server backs up files or filegroups, it:

- Backs up only the database files that you specify in the FILE or FILEGROUP option.
- Allows you to back up specific database files instead of the entire database.

When you perform database file or filegroup backups, you:

- Must specify the logical files or filegroups.
- Must perform transaction log backups in order to make restored files consistent with the rest of the database.
- Should establish a plan to back up each file on a rotating basis in order to ensure that all database files or filegroups are backed up regularly.
- Can specify up to 16 files or filegroups.

Partial Syntax

BACKUP DATABASE {*database_name* | *@database_name_var*}
[<*file_or_filegroup*> [**,** ...*m*]] TO <*backup_device*> [**,** …*n*]]

Where <file_or_filegroup> is:
{FILE = {*logical_file_name* | *@logical_file_name_var*}
|
FILEGROUP = {*logical_filegroup_name* | *@logical_filegroup_name_var*}
}

Example

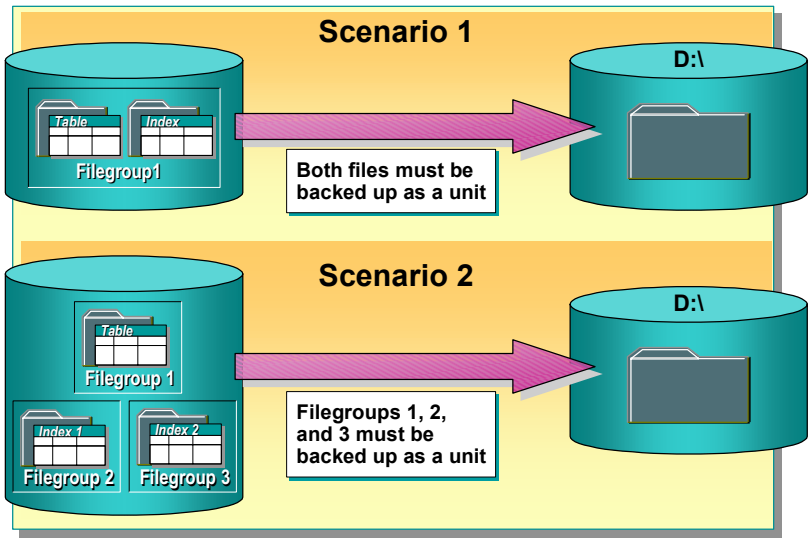
This example backs up the Orders2 file of a database filegroup. The **PhoneOrders** database consists of three files: Orders1, Orders2, and Orders3. The transaction log is stored in the Orderlog file. These backup files already exist: OrderBackup1, OrderBackup2, OrderBackup3, and OrderBackupLog.

```
BACKUP DATABASE PhoneOrders
FILE = Orders2 TO OrderBackup2
BACKUP LOG PhoneOrders to OrderBackupLog
```

Restrictions on Backing Up Database Files or Filegroups

Topic Objective
To discuss backing up indexes that are created on filegroups.

Lead-in
You may need to back up and restore several database files as a single unit if you created indexes on filegroups.



When you back up a database that consists of multiple files or filegroups, you may need to back up several database files as a single unit if you create indexes.

SQL Server automatically detects if an index was created since the last time that a database file was backed up and requires that the complete set of the affected files be backed up as a single unit.

Backing Up Indexes and Tables as a Single Unit

When you create an index in the Simple Recovery model, the transaction log records only that an index was created and the list of pages that were used to create the index. If you apply this transaction log when you restore or recover the database, SQL Server executes the CREATE INDEX statement and uses the original index pages.

In order for SQL Server to recreate the index, all database files that contain the base table, and all database files that are affected by the index creation, must be in the same condition in which they were when the index first was created.

Index and Table Are Created on the Same Filegroup

If an index and the base table are created in one filegroup, as shown in Scenario 1, you must back up the entire filegroup as a single unit.

Index and Table Are Created on Different Filegroups

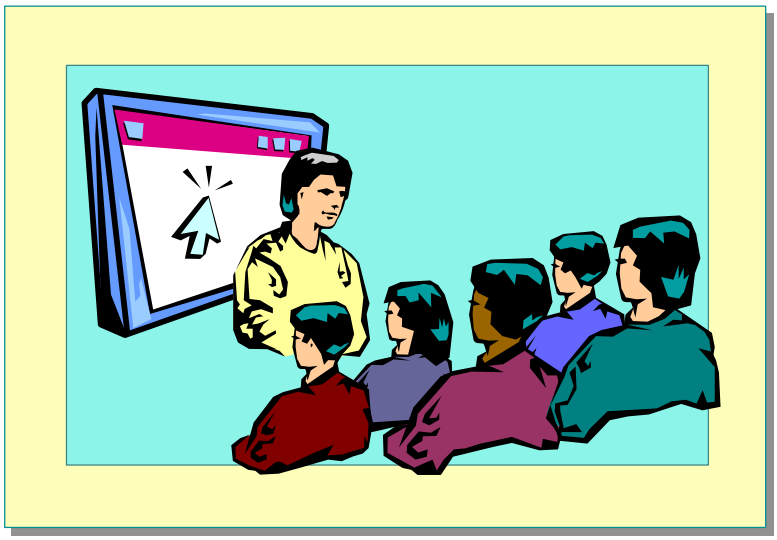
If indexes are created on multiple filegroups, and the base table is created on another filegroup, as shown in Scenario 2, you must back up all filegroups as a single unit.

For example, if the **Contact** database consists of three filegroups, where Filegroup1contains the **Customer** table, and the indexes on the **Customer** table are created in Filegroup2 and Filegroup3, you must back up all three filegroups as a single unit.

Demonstration: Using SQL Server Enterprise Manager to Perform Backups

Topic Objective
To demonstrate backing up databases with options in SQL Server Enterprise Manager.

Lead-in
In this demonstration, we'll navigate through SQL Server Enterprise Manager and see how the Transact-SQL options map to the interface.



In this demonstration, you will view how to use SQL Server Enterprise Manager to perform backups.

Planning a Backup Strategy

Topic Objective
To provide an overview of considerations for planning a backup strategy.

Lead-in
Now that you have an understanding of the different SQL Server backup methods, you can determine which method or combination of methods is appropriate for your particular business environment.

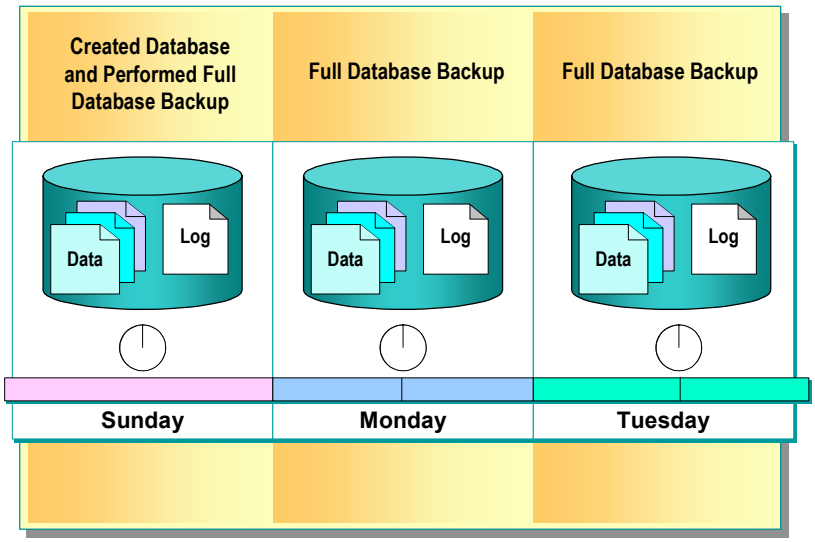
- Full Database Backup Strategy
- Full Database and Transaction Log Backup Strategy
- Differential Backup Strategy
- Database File or Filegroup Backup Strategy

When you plan a backup strategy, your particular business environment determines the backup method or combination of methods that you choose. Consider the restore process, as well as the requirements of each strategy that is presented in this module, when you determine which backup strategy to implement.

Full Database Backup Strategy

Topic Objective
To discuss the advantages of implementing a full database backup strategy and to provide examples of situations in which you would use this strategy.

Lead-in
Your database size and how frequently the data is modified determine the time and resources that are involved in implementing a full database backup strategy.



Your database size and how frequently the data is modified together determine the time and resources that are involved in implementing a full database backup strategy.

Business Implementation

Perform full database backups if:

- The database is small. The amount of time that is required to back up a small database is reasonable.
- The database has few data modifications or is read-only. Performing a full database backup captures a reasonably complete set of data. You may be willing to accept a minor loss of data if the database fails between backups and must be restored.

Transaction Log Becomes Full

If you implement only a full database backup strategy, the transaction log will eventually fill up. When the transaction log becomes full, SQL Server may prevent further database activity until you clear the transaction log:

- You should clear the transaction log periodically.
- You can set the **trunc. log on chkpt.** option to true to minimize the size of the transaction log.

> **Note** This option is provided for backward compatibility only. SQL Server 2000 uses the Simple Recovery model to implement this strategy.

When you use this option, all committed transactions are written to the database when a checkpoint occurs, and the transaction log is truncated automatically.

The transaction log does not contain the changes that were made to the database since the last full database backup.

Caution If you set the **trunc. log on chkpt.** option to true, you *cannot* back up the transaction log and use it to help restore the database in the event of a system failure.

Strategy Example 1

Consider the following example of a backup plan and the steps that you would take to restore your database. Assume the following:

- The database contains only 10 megabytes (MB) of data.
- The full database backup process takes a few minutes to complete.
- The database is used mostly for decision support and is modified very little each day.
- The possibility of losing a day's worth of changes to the database is acceptable. These changes can be recreated easily.
- The system administrator does not want to monitor the log size or perform any maintenance on the transaction log.
- The **trunc. log on chkpt.** database option is set to true to ensure that the transaction log is truncated frequently. The transaction log is not used to record changes to the database over time and cannot be used to restore the database in the event of a system failure.
- A full database backup is done each night at 6:00 P.M.
- The database becomes corrupted at 10:00 A.M.

Restore Process

To recover the database, restore the full database backup from the previous night at 6:00 P.M., overwriting the corrupted version of the database.

The limitation of this approach is that all data modifications that were made since the last full backup are lost.

Strategy Example 2

Consider the following example of a backup plan and the steps that you would take to restore your database. Assume that the database is similar to the one described in Strategy Example 1, with the following exceptions:

- The database is modified very little each day (but more frequently than the database in Strategy Example 1).
- The system administrator takes responsibility for ensuring that adequate space exists in the transaction log.
- The **trunc. log on chkpt.** database option is cleared (set to false). The transaction log records changes since the last full database backup and can be used to restore or recover the database if the system fails.

- The transaction log is stored on a separate physical device from the database.
- A full database backup is done each day at 6:00 P.M. Transaction log backups are not performed on a regular basis, but the transaction log is cleared periodically.

Restore Process

You would go through the following steps to recover the database:

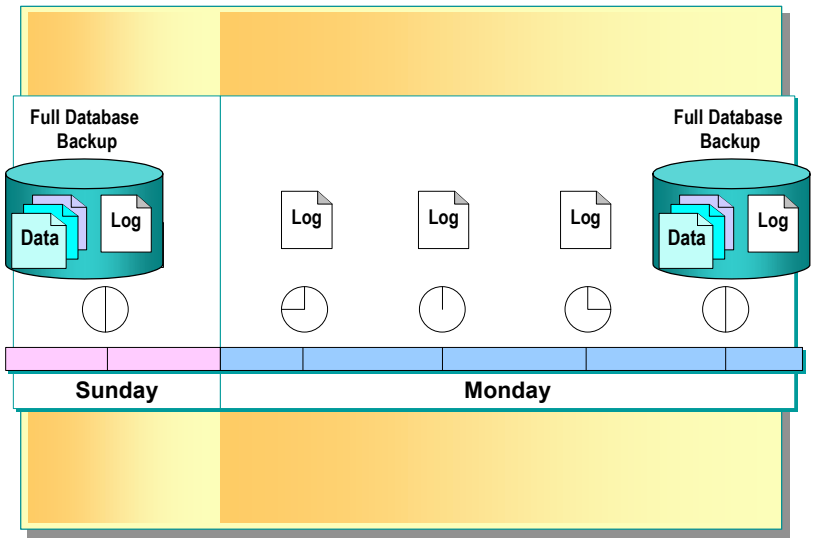
1. Back up the transaction log without truncating any data (NO_TRUNCATE option).
2. Restore the full database backup from the previous night at 6:00 P.M., overwriting the corrupted version of the database.
3. Restore the transaction log backup that you created in Step 1, and recover the database.

Using this approach, you may be able to recover changes since the backup of the previous night if the transaction log is not damaged. However, if the potential data loss is too great, you should consider implementing a backup strategy that includes periodic transaction log backups.

Full Database and Transaction Log Backup Strategy

Topic Objective
To discuss the advantages of implementing a full database and transaction log backup strategy.

Lead-in
In addition to performing a full database backup, you also should back up the transaction log.



In addition to performing a full database backup, you also should back up the transaction log in order to have a record of all database activities that occurred between full database backups. This is a common backup strategy.

When you implement a full database and transaction log backup strategy, you can restore a database from the most recent full database backup and then apply all of the transaction log backups that were created since the last full database backup.

Business Implementation

Perform a full database and transaction log backup strategy for frequently modified databases. You also should consider whether the database and transaction logs could be backed up in an acceptable amount of time.

Strategy Example

Consider the following example of a backup plan and the steps that you would take to restore your database. Assume the following:

- The database and transaction logs are stored in separate files on separate physical media.
- A full database backup is done each night at 6:00 P.M.
- The transaction log backups are performed each day at 9:00 A.M., 12:00 noon, and 3:00 P.M.
- The physical medium that contains the database is damaged at 1:30 P.M.

Restore Process

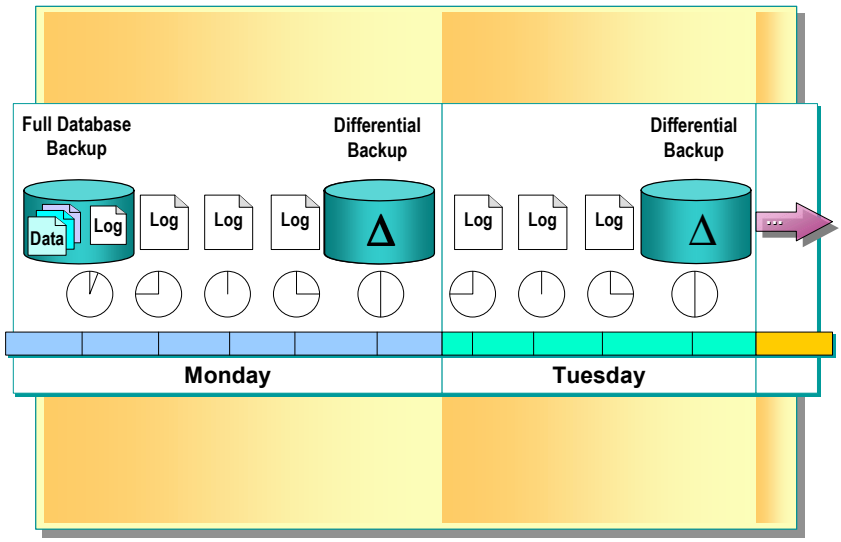
You would go through the following steps to recover the database:

1. Back up the transaction log, if possible. Use the WITH NO_TRUNCATE option.
2. Restore the full database backup that was created the previous night at 6:00 P.M.
3. Apply all transaction logs that were created that day (9:00 A.M. and 12:00 P.M.).
4. Apply the transaction log backup that was created at the beginning of the restore process (if one was created).

Differential Backup Strategy

Topic Objective
To discuss the advantages of implementing a differential with a full database and transaction log backup strategy.

Lead-in
You can perform a differential backup in addition to a full database and transaction log backup.



When you implement a differential backup strategy, you must include a full database backup, as well as transaction log backups. Differential backups consist only of the portions of the database that have changed since the last full database backup. In a differential backup, SQL Server:

- Does not capture the changes in the transaction logs. Therefore, you should back up the transaction logs periodically.
- Requires you to restore only the latest differential backup to recover a database. The latest differential backup contains all changes that were made to the database since the last full database backup.

Business Implementation

Use this strategy to reduce recovery time if the database becomes damaged. For example, rather than applying multiple, large transaction logs, you can use a differential backup to apply the changes that were made to the database since the last full database backup.

Strategy Example

Consider the following example of a backup plan and the steps that you would take to restore your database. Assume the following:

- A full database backup is performed once a week. The last full database backup was made on Sunday at 1:00 A.M.
- A differential backup is performed at the end of each business day. A differential backup was performed on both Monday and Tuesday at 6:00 P.M.
- Transaction log backups are performed every hour during the business day (8:00 A.M. to 5:00 P.M.). A transaction log backup was performed on Wednesday at 8:00 A.M. and again at 9:00 A.M.
- The database becomes corrupted on Wednesday at 9:30 A.M.

Restore Process

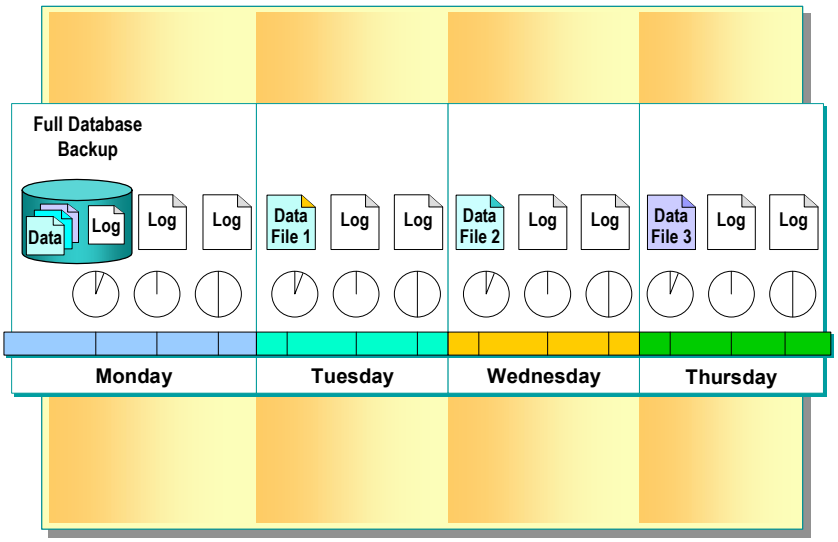
You would go through the following steps to recover the database:

1. Back up the transaction log, if possible. Use the WITH NO_TRUNCATE option.
2. Restore the full database backup that was created on Sunday at 1:00 A.M.
3. Restore the differential backup that was created on Tuesday at 6:00 P.M. This backup file is the latest differential backup and contains all changes that were made to the database since the full database backup on Sunday at 1:00 A.M.
4. Apply the transaction log backups that were created on Wednesday at 8:00 A.M. and 9:00 A.M.
5. Apply the transaction log backup that was created at the beginning of the restore process (Step 1) to ensure data consistency.

Database File or Filegroup Backup Strategy

Topic Objective
To discuss the advantages of implementing a database file and transaction log backup strategy.

Lead-in
When a database is partitioned among multiple files, you can implement a strategy that backs up selected files along with the transaction log.



When you implement a database file or filegroup backup strategy, you usually back up the transaction log as part of the strategy.

Business Implementation

Use this strategy for a VLDB that is partitioned among multiple files. When combined with regular transaction log backups, this technique offers a time-sensitive alternative to full database backups. For example, if you have only one hour to perform a full database backup (which normally takes four hours), you could back up individual files each night and still ensure data consistency.

However, this strategy is complicated and does not automatically maintain referential integrity.

Strategy Example

Consider the following example of a backup plan and the steps that you would take to restore your database. Assume the following:

- The data in a database is divided among File1, File2, and File3.
- A full database backup is performed every week. The last full database backup was performed on Monday at 1:00 A.M.
- Selected files are backed up on a rotating basis each day at 1:00 A.M:
- File1 was backed up on Tuesday at 1:00 A.M.
- File2 was backed up on Wednesday at 1:00 A.M.
- File3 was backed up on Thursday at 1:00 A.M.
- Transaction log backups are performed each day at 12:00 noon and 6:00 P.M.
- On Thursday at 8:00 A.M., the physical medium of File2 becomes damaged.

Restore Process

You would go through the following steps to recover the database:

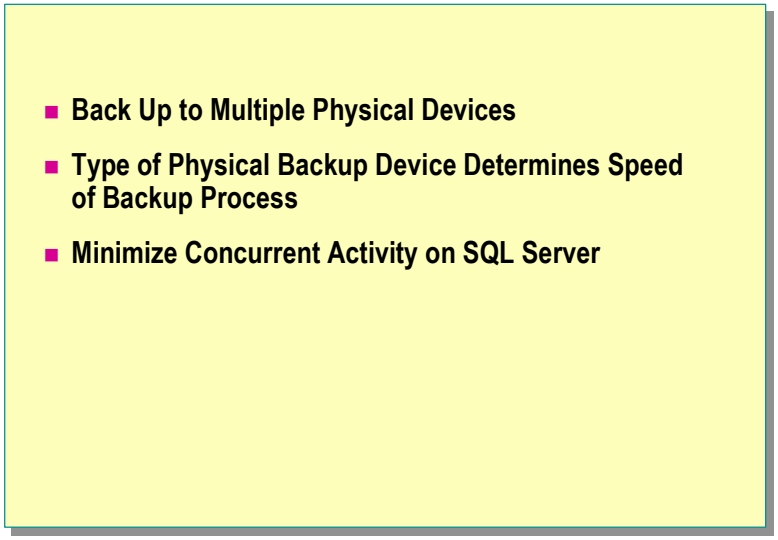
1. Back up the transaction log, if possible. Use the WITH NO_TRUNCATE option.
2. Restore the backup of File2 that was created on Wednesday at 1:00 A.M.
3. Apply all transaction log backups that were created since Wednesday at 1:00 A.M.
4. Apply the transaction log created at the beginning of the restore process to recover the data. Applying all of the transaction logs makes the objects in File2 consistent with the rest of the database.

The performance that is gained by using this strategy is a result of the fact that only transaction log events that affect data that is stored on File2 are applied. Events in the transaction log prior to 1:00 A.M. on Wednesday are not used. Only transactions for File2 after 1:00 A.M. on Wednesday are applied.

Performance Considerations

Topic Objective
To discuss performance considerations when you back up databases.

Lead-in
Consider some of the issues that impact the performance of SQL Server when you back up databases.



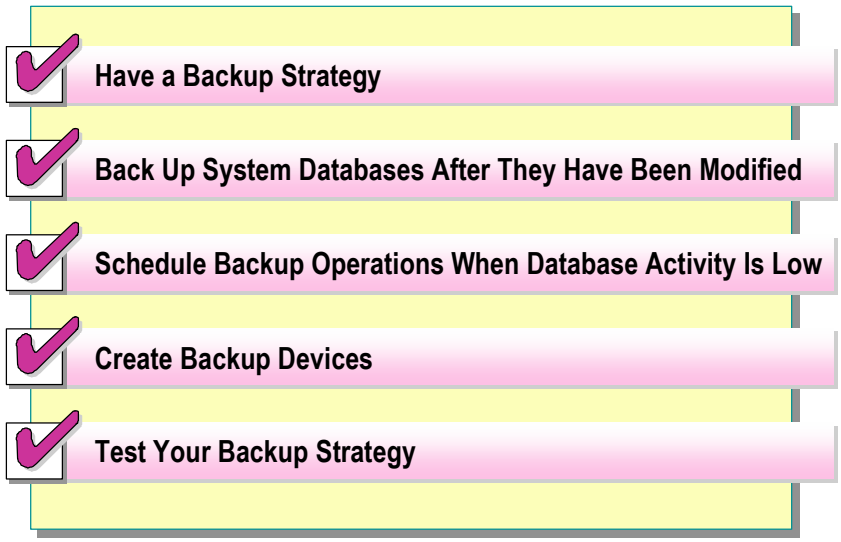
Consider some of the issues that impact the performance of SQL Server when you back up databases:

- Backing up to multiple physical devices is generally faster than using a single physical device. SQL Server takes advantage of multiple backup devices by writing the data to each backup device in parallel.
- The time that is needed to back up a database is dependent on the speed of the physical device. Tape drives are generally slower than disk devices.
- You should minimize concurrent activity when you back up a database. Concurrent activity on SQL Server may impact the time that is necessary for backing up your database.

Recommended Practices

Topic Objective
To review some of the practices that are discussed in the module in regard to backing up databases.

Lead-in
When you back up databases, consider the following practices.



The following practices should help you implement a backup strategy that is appropriate for your particular business environment and database:

- Have a backup strategy in order to minimize data loss and to recover lost data more easily. If you have a backup strategy, you can restore data with minimal cost to production time and minimize the chance of permanently lost data.
- Back up system databases after they have been modified. Remember that executing system stored procedures modifies system databases.
- Schedule backup operations when database activity is low. Although you can back up a database while the database is online and active, some operations can interfere with the backup process.
- Create backup devices so that you can reuse the backup files and automate the task of backing up your databases.
- Test your backup strategy and periodically test the data in your backup.

Additional information on the following topics is available in SQL Server Books Online.

Topic	Search on
Allowing users to back up databases	"creating user-defined SQL Server database roles"
Using Transact-SQL to create database backup files	sp_addumpdevice
Setting database options	"setting database options"
Specifying checkpoints for transaction logs	checkpoint
Backing up striped database files	striping
Backing up to a tape media	"tape media"

Lab A: Backing Up Databases

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will back up a database and transaction log with SQL Server Enterprise Manager.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Create a permanent backup file.
- Back up a database.
- Back up and clear a transaction log.
- Perform a differential backup.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2072A\Labfiles\L06.
- Answer files for this lab, which are located in C:\Moc\2072A\Labfiles\L06\Answers.

For More Information

If you require help in executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious nwtraders.msft domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.x.1
SQLAdmin2	Denver	192.168.x.2
SQLAdmin3	Perth	192.168.x.3
SQLAdmin4	Brisbane	192.168.x.4
SQLAdmin5	Lisbon	192.168.x.5
SQLAdmin6	Bonn	192.168.x.6
SQLAdmin7	Lima	192.168.x.7
SQLAdmin8	Santiago	192.168.x.8
SQLAdmin9	Bangalore	192.168.x.9
SQLAdmin10	Singapore	192.168.x.10
SQLAdmin11	Casablanca	192.168.x.11
SQLAdmin12	Tunis	192.168.x.12
SQLAdmin13	Acapulco	192.168.x.13
SQLAdmin14	Miami	192.168.x.14
SQLAdmin15	Auckland	192.168.x.15
SQLAdmin16	Suva	192.168.x.16
SQLAdmin17	Stockholm	192.168.x.17
SQLAdmin18	Moscow	192.168.x.18
SQLAdmin19	Caracas	192.168.x.19
SQLAdmin20	Montevideo	192.168.x.20
SQLAdmin21	Manila	192.168.x.21
SQLAdmin22	Tokyo	192.168.x.22
SQLAdmin23	Khartoum	192.168.x.23
SQLAdmin24	Nairobi	192.168.x.24

Estimated time to complete this lab: 45 minutes

Exercise 1
Creating Backup Devices

In this exercise, you will use SQL Server Enterprise Manager and Transact-SQL statements to create backup devices to contain database and transaction log backups.

► To create a backup device using SQL Server Enterprise Manager

In this procedure, you will use SQL Server Enterprise Manager to create two backup devices. Labfiles\L06\Answers\MakeDev1.sql is a completed script for this procedure.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the nwtraders.msft classroom domain)
Password	**password**

2. Open SQL Server Enterprise Manager.
3. In the console tree, expand **Microsoft SQL Servers**, and then expand **SQL Server Group**.
4. Expand your server, expand **Management**, right-click **Backup**, and then click **New Backup Device**.
5. Using Windows Explorer, create a folder named Backup in the root directory of drive C.
6. Use the information in the following table to create two permanent backup files with SQL Server Enterprise Manager.

Name	Disk file name
Nw1	C:\Backup\Nw1.bak
Nwlog	C:\ Backup\Nwlog.bak

7. Close SQL Server Enterprise Manager.

► To create backup devices using Transact-SQL

In this procedure, you will create two backup devices by using the **sp_addumpdevice** system stored procedure. Labfiles\L06\Answers\MakeDev2.sql is a completed script for this procedure.

1. Open SQL Query Analyzer and, if requested, log in to the (local) server with Windows Authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

2. Write and execute a Transact-SQL statement to create two permanent backup files based on the information in the following table.

Device type	Logical name	Physical name
Disk	Nwstripe1	C:\ Backup\Nwstripe1.bak
Disk	Nwstripe2	C:\ Backup\Nwstripe2.bak

► To confirm that new devices were created

In this procedure, you will use SQL Server Enterprise Manager to confirm that Nwstripe1 and Nwstripe2 were created.

1. Open SQL Server Enterprise Manager.
2. In the console tree, expand **Microsoft SQL Servers**, and then expand **SQL Server Group**.
3. Expand your server, expand **Management**, right-click **Backup**, and then click **Refresh**.
4. Confirm that the Nwstripe1 and Nwstripe2 devices were created.

Exercise 2
Backing Up Databases

In this exercise, you will perform several full database backups using SQL Server Enterprise Manager and Transact-SQL statements.

► To clear the database options

In this procedure, you will clear any database options that are set for the **Northwind** database.

1. In the console tree, expand your server, and then click **Databases**.
2. In the details pane, right-click **Northwind**, and then click **Properties**.
3. On the Options tab, clear any selected options.
4. Click **OK**.

► To back up a database using SQL Server Enterprise Manager

In this procedure, you will perform a full database backup of the **Northwind** database to the Nw1 backup file on disk.

1. In the console tree, right-click **Backup**, and then click **Backup a database**.
2. In the SQL Server Backup window, fill in the options using the information in the following table.

Field	Value
Database	**Northwind**
Name	NorthwindFull
Description	The first backup of **Northwind**
Backup type	Database—complete
Destination	Nw1
Overwrite	Overwrite existing media

3. Click **OK** to perform the backup.

► To append subsequent backups to one backup file using Transact-SQL

In this procedure, you will write and execute a Transact-SQL statement to perform another full database backup of the **Northwind** database and append the backup to the Nw1 backup file. Labfiles\L06\Answers\Append.sql is a completed script for this procedure.

1. Switch to SQL Query Analyzer.
2. Write and execute a Transact-SQL statement that backs up the **Northwind** database and appends the backup to the Nw1 backup file. Use the options in the following table.

Option	Value
Database name	**Northwind**
Backup file (logical device)	Nw1
Append, overwrite, or initialize	Append(NOINIT)
Description	The second full backup of **Northwind**

► To view the contents of a backup device

In this procedure, you will use SQL Server Enterprise Manager to view the contents of the Nw1 backup file to ensure that it contains two complete database backups.

1. Switch to SQL Server Enterprise Manager.
2. In the console tree, expand **Management**, and then click **Backup**.
3. In the details pane, right-click **Nw1**, and then click **Properties**.
4. Click **View Contents**.
5. Confirm that the backup file contains two complete database backups.

► To overwrite an existing backup using Transact-SQL

In this procedure, you will write and execute a Transact-SQL statement to back up the **Northwind** database and overwrite any existing backups on the Nw1 backup file. Labfiles\L06\Answers\Overwrite.sql is a completed script for this procedure.

1. Switch to SQL Query Analyzer.
2. Write and execute a Transact-SQL statement to back up the **Northwind** database onto the Nw1 backup file. Use the information in the following table.

Option	Value
Database name	**Northwind**
Backup file (logical device)	Nw1
Append, overwrite, or initialize	Overwrite(INIT)
Description	A third backup of **Northwind** overwrites all others

► To back up one database to multiple backup files

In this procedure, you will use SQL Server Enterprise Manager to perform a full database backup of the **Northwind** database onto two existing backup files: Nwstripe1 and Nwstripe2. You will also overwrite any existing data, including header information.

1. Switch to SQL Server Enterprise Manager.
2. In the console tree, right-click **Backup**, and then click **Backup a database**.
3. In the SQL Server Backup window, fill in the options using the information in the following table.

Field	Value
Database	**Northwind**
Name	Northwind striped
Description	A parallel backup of **Northwind**
Backup type	Database—complete
Destination	Nwstripe1 and Nwstripe2 (If the Nw1 backup file is the Destination list, remove it.)
Overwrite	Overwrite existing media

4. Click **OK** to perform the backup.

What do the Nwstripe1 and Nwstripe2 backup files contain?

The content of Nwstripe1 is Family 1, Media 1. The content of Nwstripe2 is Family 2, Media 1.

_____

_____

► To back up a database and create a temporary backup device

In this procedure, you will write and execute a single Transact-SQL statement to back up the **Northwind** database to a new, temporary backup file. Labfiles\L06\Answers\BacToTmp.sql is a completed script for this procedure.

1. Switch to SQL Query Analyzer.
2. Write and execute a single Transact-SQL statement that backs up the **Northwind** database to a new, temporary backup file. Use the information in the following table.

Option	Value
Database name	**Northwind**
File location	C:\ Backup\MyNewBackup.bak
Append, overwrite, or initialize	Initialize(FORMAT)
Description	New temporary backup device, not recorded in system tables

3. Start Windows Explorer.
4. Expand **C: \Backup** to confirm that the MyNewBackup.bak file was created and populated.
5. Switch to SQL Server Enterprise Manager.
6. In the console tree, right-click **Northwind**, point to **All Tasks**, and then click **Backup Database**.
7. In the **SQL Server Backup** dialog box, in the **Destination** list, click **MyNewBackup.bak** and then click **Contents** to view information about the contents of the temporary device.

Exercise 3
Backing Up a Transaction Log

In this exercise, you will back up a transaction log to capture changes to the database. You will use SQL Server Enterprise Manager, as well as Transact-SQL statements, to back up a transaction log.

► To back up a transaction log using SQL Server Enterprise Manager

In this procedure, you will use SQL Server Enterprise Manager to back up the transaction log for the **Northwind** database onto the Nwlog backup file.

1. In the console tree, right-click **Northwind**, and then click **Properties**.
2. On the **Options** tab, change the Recovery Model to **Full**, and then click **OK**.
3. In the console tree, right-click **Backup**, and then click **Backup a database**.
4. In the SQL Server Backup window, fill in the options using the information in the following table.

Field	Value
Database	**Northwind**
Name	NwindLog
Description	Northwind transaction log
Backup type	Transaction log
Destination	Nwlog
Overwrite	Overwrite existing media

5. Click **OK** to perform the backup.
6. After the backup completes, review the contents of the Nwlog backup file with SQL Server Enterprise Manager.

► To back up a transaction log using Transact-SQL statements

In this procedure, you will write and execute a Transact-SQL statement to append a second backup of the transaction log onto the Nwlog backup file. Labfiles\L06\Answers\AppendLg.sql is a completed script for this procedure.

1. Switch to SQL Query Analyzer.
2. Write and execute a Transact-SQL statement to append a second backup of the transaction log onto the Nwlog backup file. Use the information in the following table.

Option	Value
Database name	**Northwind**
Backup file (logical device)	Nwlog
Append, overwrite, or initialize	Append(NOINIT)

► To clear a transaction log without making a backup copy

In this procedure, you will write and execute a Transact-SQL statement to clear the transaction log of the **Northwind** database. Assume that the log has become full. After you clear the transaction log, you must back up the database. Labfiles\L06\Answers\ClearLog.sql is a completed script for this procedure.

1. Write and execute a Transact-SQL statement to clear the transaction log of all committed transactions.
2. Write and execute a Transact-SQL statement to back up the database to the Nw1 backup file.

Exercise 4
Performing a Differential Backup

In this exercise, you will perform a differential backup to capture the latest changes to all rows in the **Northwind** database with SQL Server Enterprise Manager or Transact-SQL statements.

► To perform a differential backup

In this procedure, you will perform a differential backup of the **Northwind** database and append the differential backup to the Nwdiff.bak backup file. Labfiles\L06\Answers\DiffBac.sql is a completed script for this procedure.

1. In the console tree, right-click **Backup**, and then click **Backup a database**.
2. In the SQL Server Backup window, fill in the options using the information in the following table.

Field	Value
Database	**Northwind**
Name	Nwind differential
Description	Changes since the last full database backup
Backup type	Database—differential
Destination	C:\Backup\Nwdiff.bak (Enter this in the **File name** field)
Overwrite	Append to media

3. Click **OK** to perform the backup.
4. After the backup operation completes, right-click **Northwind**, point to **All Tasks**, and then click **Backup Database**.
5. In the **Backup Database** dialog box, in the **Destination** list, click **Nwdiff.bak**, and then click **Contents** to view information about the contents of the temporary device.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Preventing Data Loss
- Setting and Changing a Database Recovery Model
- SQL Server Backup
- When to Back Up Databases
- Performing Backups
- Types of Backup Methods
- Planning a Backup Strategy

1. Your database consists of 5 gigabytes (GB) of data and is stored in one database file. This database is used as an order-taking system for a mail-order catalog company. Operators take orders 24 hours a day. The company typically receives about 12,000 orders each day. Describe an appropriate backup plan for this database.

   SQL Server backups can occur while the database is online. However, avoid scheduling backups during high database activity.

   Because the database exists on a single database file, you cannot back up individual parts of the database. You must back up the entire database as a single unit.

   Consider a backup plan that includes full database and transaction log backups. You may want to add differential backups as the volume of daily orders increases. These differential backups shorten the restore time if the system fails.

2. Your database contains image data that is gathered from a weather satellite and is being continually updated. The database is 700 GB. The database is partitioned onto three files. If you were to perform a full database backup, the process would take about 20 hours. How can you minimize the amount of time that is spent performing backups each day and yet still ensure good data recoverability in the event of a system failure?

   Use a backup plan that starts with one full database backup. A full database backup will be done infrequently. Perform a backup of one of the database files each day on a rotating basis. Perform differential backups in addition to transaction log backups in order to minimize recovery time.

3. You have a database for which you generally perform *only* full database backups. The transaction log exists on a separate physical disk from the data files. It is allowed to accumulate changes but is periodically cleared. The disk that contains the data files is damaged. After you replace the disk, what can you do to minimize data loss?

   Try to back up the undamaged transaction log by using the NO_TRUNCATE option. This captures some of the activity since the last full database backup. After you restore the database, apply the transaction log backup and recover the database.

4. What are the advantages and disadvantages of using differential backups as part of your backup strategy?

   Differential backups save time in the restore process. You can recover a database by restoring the full backup and the last differential backup only. It is not necessary to apply all of the transaction logs or previous differential backups in order to bring the database to a consistent state.

   A disadvantage of differential backups is that because differential backups do not capture intermediate changes to the database, you cannot use them to recover data from a specific point in time. You must use transaction log backups to perform point-in-time recovery. In addition, each new differential backup will be larger than the previous one, as the time increases between the last full database backup and the differential backup.