

Backup and Restore of SQL Server Databases

SQL Server 2012 Books Online

Reference



Microsoft®

Backup and Restore of SQL Server Databases

SQL Server 2012 Books Online

Summary: This book describes the benefits of backing up SQL Server databases, basic backup and restore terms, and introduces backup and restore strategies for SQL Server and security considerations for SQL Server backup and restore.

Category: Reference

Applies to: SQL Server 2012

Source: SQL Server Books Online ([link to source content](#))

E-book publication date: June 2012

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at

<http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Contents

Backup and Restore of SQL Server Databases.....	6
Recovery Models.....	13
View or Change the Recovery Model of a Database.....	16
Backup Overview.....	19
Restore and Recovery Overview.....	23
Plan and Perform Restore Sequences (Full Recovery Model).....	30
Restart an Interrupted Restore Operation (Transact-SQL).....	32
Backup Compression.....	33
Configure Backup Compression.....	35
Use Resource Governor to Limit CPU Usage by Backup Compression (Transact-SQL).....	36
Full Database Backups.....	43
Create a Full Database Backup.....	46
Back Up Database (General Page).....	54
Back Up Database (Options Page).....	57
Select Backup Destination.....	61
Partial Backups.....	63
Full File Backups.....	64
Back Up Files and Filegroups.....	67
Differential Backups.....	73
Create a Differential Database Backup.....	76
Restore a Differential Database Backup.....	81
Copy-Only Backups.....	86
Transaction Log Backups.....	87
Back Up a Transaction Log.....	89
Tail-Log Backups.....	93
Back Up the Transaction Log When the Database Is Damaged.....	96
Backup Devices.....	100
Define a Logical Backup Device for a Disk File.....	108
Define a Logical Backup Device for a Tape Drive.....	110
View the Contents of a Backup Tape or File.....	112
Specify a Disk or Tape As a Backup Destination.....	114
Device Contents.....	115
Backup Device (Media Contents Page).....	117
Backup Device (General Page).....	119

Restore a Backup from a Device.....	121
Delete a Backup Device	123
Media Sets, Media Families, and Backup Sets.....	125
Set the Expiration Date on a Backup	134
View the Data and Log Files in a Backup Set.....	136
Mirrored Backup Media Sets	138
Back Up to a Mirrored Media Set (Transact-SQL).....	140
Backup History and Header Information	141
View the Properties and Contents of a Logical Backup Device.....	148
Possible Media Errors During Backup and Restore.....	150
Enable or Disable Backup Checksums During Backup or Restore.....	152
Specify Whether a Backup or Restore Operation Continues or Stops After Encountering an Error.....	155
Complete Database Restores (Simple Recovery Model).....	157
Restore a Database Backup Under the Simple Recovery Model (Transact-SQL).....	161
Restore a Database Backup (SQL Server Management Studio)	163
Backup Timeline.....	167
Restore Database (General Page).....	168
Restore Database (Options Page).....	175
Restore Database (Files Page)	178
Continue with Restore	179
Select Backup Device	180
Restore a Database to a New Location.....	180
Complete Database Restores (Full Recovery Model)	187
Restore a Database to the Point of Failure Under the Full Recovery Model (Transact-SQL) ...	192
File Restores (Simple Recovery Model).....	195
Restore Files and Filegroups over Existing Files	197
Restore Files to a New Location	201
Restore Files and Filegroups.....	206
File Restores (Full Recovery Model)	212
Apply Transaction Log Backups.....	216
Restore a Transaction Log Backup.....	219
Restore a SQL Server Database to a Point in Time (Full Recovery Model).....	227
Recovery of Related Databases That Contain Marked Transaction.....	232
Use Marked Transactions to Recover Related Databases Consistently (Full Recovery Model)	234
Restore a Database to a Marked Transaction (SQL Server Management Studio).....	239
Recover to a Log Sequence Number.....	240
Online Restore.....	242

Deferred Transactions	245
Remove Defunct Filegroups.....	247
Example: Online Restore of a Read/Write File (Full Recovery Model).....	250
Example: Online Restore of a Read-Only File (Full Recovery Model).....	251
Example: Online Restore of a Read-Only File (Simple Recovery Model)	252
Example: Offline Restore of Primary and One Other Filegroup (Full Recovery Model).....	253
Restore Pages.....	254
Manage the suspect_pages Table.....	262
Piecemeal Restores.....	266
Example: Piecemeal Restore of Database (Full Recovery Model)	271
Example: Piecemeal Restore of Database (Simple Recovery Model).....	272
Example: Piecemeal Restore of Only Some Filegroups (Full Recovery Model).....	274
Example: Piecemeal Restore of Only Some Filegroups (Simple Recovery Model).....	275
Recover a Database Without Restoring Data (Transact-SQL)	277
Back Up and Restore of System Databases.....	279
Restore the master Database (Transact-SQL).....	283
Backup and Restore: Interoperability and Coexistence	284

Backup and Restore of SQL Server Databases

This topic describes the benefits of backing up SQL Server databases, basic backup and restore terms, and introduces backup and restore strategies for SQL Server and security considerations for SQL Server backup and restore.

The SQL Server backup and restore component provides an essential safeguard for protecting critical data stored in your SQL Server databases. To minimize the risk of catastrophic data loss, you need to back up your databases to preserve modifications to your data on a regular basis. A well-planned backup and restore strategy helps protect databases against data loss caused by a variety of failures. Test your strategy by restoring a set of backups and then recovering your database to prepare you to respond effectively to a disaster.

In this Topic:

- Benefits
- Components and Concepts
- Introduction to Backup and Restore Strategies
- Related Tasks

Benefits

- Backing up your SQL Server databases, running test restores procedures on your backups, and storing copies of backups in a safe, off-site location protects you from potentially catastrophic data loss.

Important

This is the only way to reliably protect your SQL Server data.

With valid backups of a database, you can recover your data from many failures, such as:

- Media failure.
 - User errors, for example, dropping a table by mistake.
 - Hardware failures, for example, a damaged disk drive or permanent loss of a server.
 - Natural disasters.
- Additionally, backups of a database are useful for routine administrative purposes, such as copying a database from one server to another, setting up AlwaysOn Availability Groups or database mirroring, and archiving.

Components and Concepts

back up [verb]

Copies the data or log records from a SQL Server database or its transaction log to a backup device, such as a disk, to create a data backup or log backup.

backup [noun]

A copy of data that can be used to restore and recover the data after a failure. Backups of a database can also be used to restore a copy the database to a new location.

backup device

A disk or tape device to which SQL Server backups are written and from which they can be restored.

backup media

One or more tapes or disk files to which one or more backup have been written.

data backup

A backup of data in a complete database (a database backup), a partial database (a partial backup), or a set of data files or filegroups (a file backup).

database backup

A backup of a database. Full database backups represent the whole database at the time the backup finished. Differential database backups contain only changes made to the database since its most recent full database backup.

differential backup

A data backup that is based on the latest full backup of a complete or partial database or a set of data files or filegroups (the differential base) and that contains only the data that has changed since that base.

full backup

A data backup that contains all the data in a specific database or set of filegroups or files, and also enough log to allow for recovering that data.

log backup

A backup of transaction logs that includes all log records that were not backed up in a previous log backup. (full recovery model)

recover

To return a database to a stable and consistent state.

recovery

A phase of database startup or of a restore with recovery that brings the database into a transaction-consistent state.

recovery model

A database property that controls transaction log maintenance on a database. Three recovery models exist: simple, full, and bulk-logged. The recovery model of database determines its backup and restore requirements.

restore

A multi-phase process that copies all the data and log pages from a specified SQL Server backup to a specified database, and then rolls forward all the transactions that are logged in the backup by applying logged changes to bring the data forward in time.

Introduction to Backup and Restore Strategies

Backing up and restoring data must be customized to a particular environment and must work with the available resources. Therefore, a reliable use of backup and restore for recovery requires a backup and restore strategy. A well-designed backup and restore strategy maximizes data availability and minimizes data loss, while considering your particular business requirements.

Important

Place the database and backups on separate devices. Otherwise, if the device containing the database fails, your backups will be unavailable. Placing the data and backups on separate devices also enhances the I/O performance for both writing backups and the production use of the database.

A backup and restore strategy contains a backup portion and a restore portion. The backup part of the strategy defines the type and frequency of backups, the nature and speed of the hardware that is required for them, how backups are to be tested, and where and how backup media is to be stored (including security considerations). The restore part of the strategy defines who is responsible for performing restores and how restores should be performed to meet your goals for availability of the database and for minimizing data loss. We recommend that you document your backup and restore procedures and keep a copy of the documentation in your run book.

Designing an effective backup and restore strategy requires careful planning, implementation, and testing. Testing is required. You do not have a backup strategy until you have successfully restored backups in all the combinations that are included in your restore strategy. You must consider a variety of factors. These include the following:

- The production goals of your organization for the databases, especially the requirements for availability and protection of data from loss.
- The nature of each of your databases: its size, its usage patterns, the nature of its content, the requirements for its data, and so on.
- Constraints on resources, such as: hardware, personnel, space for storing backup media, the physical security of the stored media, and so on.



Note

The SQL Server on-disk storage format is the same in the 64-bit and 32-bit environments. Therefore, backup and restore work across 32-bit and 64-bit environments. A backup created on a server instance running in one environment can be restored on a server instance that runs in the other environment.

Impact of the Recovery Model on Backup and Restore

Backup and restore operations occur within the context of a recovery model. A recovery model is a database property that controls how the transaction log is managed. Also, the recovery model of a database determines what types of backups and what restore scenarios are supported for the database. Typically a database uses either the simple recovery model or the full recovery model. The full recovery model can be supplemented by switching to the bulk-logged recovery model before bulk operations. For an introduction to these recovery models and how they affect transaction log management, see [Transaction Logs \(SQL Server\)](#).

The best choice of recovery model for the database depends on your business requirements. To avoid transaction log management and simplify backup and restore, use the simple recovery model. To minimize work-loss exposure, at the cost of administrative overhead, use the full recovery model. For information about the effect of recovery models on backup and restore, see [Backup Overview \(SQL Server\)](#).

Design the Backup Strategy

After you have selected a recovery model that meets your business requirements for a specific database, you have to plan and implement a corresponding backup strategy. The optimal backup strategy depends on a variety of factors, of which the following are especially significant:

- How many hours a day do applications have to access the database?
If there is a predictable off-peak period, we recommend that you schedule full database backups for that period.
- How frequently are changes and updates likely to occur?
If changes are frequent, consider the following:
 - Under the simple recovery model, consider scheduling differential backups between full database backups. A differential backup captures only the changes since the last full database backup.
 - Under the full recovery model, you should schedule frequent log backups. Scheduling differential backups between full backups can reduce restore time by reducing the number of log backups you have to restore after restoring the data.

- Are changes likely to occur in only a small part of the database or in a large part of the database?

For a large database in which changes are concentrated in a part of the files or filegroups, partial backups and or file backups can be useful. For more information, see [Partial Backups](#) and [Full File Backups](#).

- How much disk space will a full database backup require?

For more information, see Estimating the Size of a Full Database Backup, later in this section.

Estimate the Size of a Full Database Backup

Before you implement a backup and restore strategy, you should estimate how much disk space a full database backup will use. The backup operation copies the data in the database to the backup file. The backup contains only the actual data in the database and not any unused space. Therefore, the backup is usually smaller than the database itself. You can estimate the size of a full database backup by using the **sp_spaceused** system stored procedure. For more information, see [sp_spaceused \(Transact-SQL\)](#).

Schedule Backups

Performing a backup operation has minimal effect on transactions that are running; therefore, backup operations can be run during regular operations. You can perform a SQL Server backup with minimal effect on production workloads.



Note

For information about concurrency restrictions during backup, see [Backup Overview \(SQL Server\)](#).

After you decide what types of backups you require and how frequently you have to perform each type, we recommend that you schedule regular backups as part of a database maintenance plan for the database. For information about maintenance plans and how to create them for database backups and log backups, see [Use the Maintenance Plan Wizard](#).

Test Your Backups

You do not have a restore strategy until you have tested your backups. It is very important to thoroughly test your backup strategy for each of your databases by restoring a copy of the database onto a test system. You must test restoring every type of backup that you intend to use.

We recommend that you maintain an operations manual for each database. This operations manual should document the location of the backups, backup device names (if any), and the amount of time that is required to restore the test backups.

Related Tasks

Scheduling Backup Jobs

- [How to: Create a Maintenance Plan](#)
- [How to: Create a Job \(SQL Server Management Studio\)](#)
- [How to: Schedule a Job \(SQL Server Management Studio\)](#)

Working with Backup Devices and Backup Media

- [How to: Define a Logical Backup Device for a Disk File \(SQL Server Management Studio\)](#)
- [How to: Define a Logical Backup Device for a Tape Drive \(SQL Server Management Studio\)](#)
- [How to: Specify a Disk or Tape as a Backup Destination \(SQL Server Management Studio\)](#)
- [How to: Delete a Backup Device \(SQL Server Management Studio\)](#)
- [How to: Set the Expiration Date on a Backup \(SQL Server Management Studio\)](#)
- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)
- [How to: View the Data and Log Files In a Backup Set \(SQL Server Management Studio\)](#)
- [How to: View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)
- [How to: Restore a Backup From a Device \(SQL Server Management Studio\)](#)

Creating Backups



Note

For partial or copy-only backups, you must use the Transact-SQL [BACKUP](#) statement with the PARTIAL or COPY_ONLY option, respectively.

Using SQL Server Management Studio

- [How to: Back Up a Database \(SQL Server Management Studio\)](#)
- [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)
- [How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)
- [How to: Create a Differential Database Backup \(SQL Server Management Studio\)](#)

Using Transact-SQL

- [How to: Use Resource Governor to Limit CPU Usage by Backup Compression \(Transact-SQL\)](#)
- [How to: Back Up the Transaction Log When the Database Is Damaged \(Transact-SQL\)](#)
- [How to: Enable or Disable the Backup CHECKSUM Option \(Transact-SQL\)](#)

- [How to: Handle CHECKSUM Errors During Backup \(Transact-SQL\)](#)

Restoring Data Backups

Using SQL Server Management Studio

- [How to: Restore a Database Backup \(SQL Server Management Studio\)](#)
- [How to: Create a New Database From An Existing Database Backup \(SQL Server Management Studio\)](#)
- [How to: Restore a Differential Backup \(SQL Server Management Studio\)](#)
- [How to: Restore Files and Filegroups \(SQL Server Management Studio\)](#)

Using Transact-SQL

- [How to: Restore a Database Backup Under the Simple Recovery Model \(Transact-SQL\)](#)
- [How to: Restore a Database to the Point of Failure Under the Full Recovery Model \(Transact-SQL\)](#)
- [How to: Restore Files and Filegroups over Existing Files \(Transact-SQL\)](#)
- [How to: Restore Files to a New Location \(Transact-SQL\)](#)
- [How to: Restore the master Database \(Transact-SQL\)](#)

Restoring Transaction Logs (Full Recovery Model)

Using SQL Server Management Studio

- [How to: Restore a Database to a Marked Transaction \(SQL Server Management Studio\)](#)
- [How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)
- [Restore a Database to a Point In Time \(SQL Server\)](#)

Using Transact-SQL

- [Restore a Database to a Point In Time \(SQL Server\)](#)

Additional Restore Tasks

Using Transact-SQL

- [How to: Restart an Interrupted Restore Operation \(Transact-SQL\)](#)
- [How to: Recover a Database from a Backup Without Restoring Data \(Transact-SQL\)](#)

See Also

[Backup Overview \(SQL Server\)](#)

[Restore and Recovery Overview \(SQL Server\)](#)

[BACKUP \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[Backing Up and Restoring an Analysis Services Database](#)

[Backing up and Restoring Full-Text Catalogs](#)

[Backing Up and Restoring Replicated Databases](#)

[Transaction Logs \(SQL Server\)](#)

[Recovery Models \(SQL Server\)](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

Recovery Models

SQL Server backup and restore operations occur within the context of the recovery model of the database. Recovery models are designed to control transaction log maintenance. A *recovery model* is a database property that controls how transactions are logged, whether the transaction log requires (and allows) backing up, and what kinds of restore operations are available. Three recovery models exist: simple, full, and bulk-logged. Typically, a database uses the full recovery model or simple recovery model. A database can be switched to another recovery model at any time.

In this Topic:

- [Recovery Model Overview](#)
- [Related Tasks](#)

Recovery Model Overview

The following table summarizes the three recovery models.

Recovery model	Description	Work loss exposure	Recover to point in time?
Simple	No log backups. Automatically reclaims log space to keep space requirements small, essentially eliminating the need to manage the	Changes since the most recent backup are unprotected. In the event of a disaster, those changes must be redone.	Can recover only to the end of a backup. For more information, see Complete Database Restore (Simple Recovery Model) .

Recovery model	Description	Work loss exposure	Recover to point in time?
	<p>transaction log space. For information about database backups under the simple recovery model, see Full Database Backups (SQL Server).</p>		
Full	<p>Requires log backups. No work is lost due to a lost or damaged data file. Can recover to an arbitrary point in time (for example, prior to application or user error). For information about database backups under the full recovery model, see Full Database Backups (SQL Server) and Complete Database Restores (Full Recovery Model).</p>	<p>Normally none. If the tail of the log is damaged, changes since the most recent log backup must be redone.</p>	<p>Can recover to a specific point in time, assuming that your backups are complete up to that point in time. For information about using log backups to restore to the point of failure, see Restore a SQL Server Database to a Point in Time (Full Recovery Model).</p> <p> Note If you have two or more full-recovery-model databases that must be logically consistent, you may have to implement special procedures to make sure the recoverability of these databases. For more information, see Recovery of Related Databases That Contain Marked Transaction.</p>

Recovery model	Description	Work loss exposure	Recover to point in time?
Bulk logged	<p>Requires log backups. An adjunct of the full recovery model that permits high-performance bulk copy operations. Reduces log space usage by using minimal logging for most bulk operations. For information about operations that can be minimally logged, see Transaction Logs (SQL Server). For information about database backups under the bulk-logged recovery model, see Full Database Backups (SQL Server) and Complete Database Restores (Full Recovery Model).</p>	<p>If the log is damaged or bulk-logged operations occurred since the most recent log backup, changes since that last backup must be redone. Otherwise, no work is lost.</p>	<p>Can recover to the end of any backup. Point-in-time recovery is not supported.</p>

Related Tasks

- [View or Change the Recovery Model of a Database \(SQL Server\)](#)
- [Troubleshoot a Full Transaction Log \(Error 9002\)](#)

See Also

- [backupset \(Transact-SQL\)](#)
- [sys.databases \(Transact-SQL\)](#)
- [ALTER DATABASE SET Options \(Transact-SQL\)](#)
- [Backing Up and Restoring Databases](#)

[Transaction Logs](#)

[Automating Administrative Tasks \(SQL Server Agent\)](#)

[Restore and Recovery Overview \(SQL Server\)](#)

View or Change the Recovery Model of a Database

This topic describes how to view or change the recovery model of a database in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. A *recovery model* is a database property that controls how transactions are logged, whether the transaction log requires (and allows) backing up, and what kinds of restore operations are available. Three recovery models exist: simple, full, and bulk-logged. Typically, a database uses the full recovery model or simple recovery model. A database can be switched to another recovery model at any time. The **model** database sets the default recovery model of new databases.

In This Topic

- **Before you begin:**
 - Recommendations
 - Security
- **To view or change the recovery model of a database, using:**
 - SQL Server Management Studio
 - Transact-SQL
- **Follow Up Recommendations:** After You Change the Recovery Model
- Related Tasks

Before You Begin

Recommendations

- Before switching from the full recovery or bulk-logged recovery model, back up the transaction log.
- Point-in-time recovery is not possible with bulk-logged model. Therefore, if you run transactions under the bulk-logged recovery model that might require a transaction log restore, these transactions could be exposed to data loss. To maximize data recoverability in a disaster-recovery scenario, we recommend that you switch to the bulk-logged recovery model only under the following conditions:
 - Users are currently not allowed in the database.
 - All modifications made during bulk processing are recoverable without depending on taking a log backup; for example, by re-running the bulk processes.

If you satisfy these two conditions, you will not be exposed to any data loss while restoring a transaction log that was backed up under the bulk-logged recovery model..



Note

If you switch to the full recovery model during a bulk operation, the logging of the bulk operation changes from minimal logging to full logging, and vice versa.

Security

Permissions

Requires ALTER permission on the database.



Using SQL Server Management Studio

▶ To view or change the recovery model

1. After connecting to the appropriate instance of the SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, and then click **Properties**, which opens the **Database Properties** dialog box.
4. In the **Select a page** pane, click **Options**.
5. The current recovery model is displayed in the **Recovery model** list box.
6. Optionally, to change the recovery model select a different model list. The choices are **Full**, **Bulk-logged**, or **Simple**.
7. Click .



Using Transact-SQL

▶ To view the recovery model

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. This example shows how to query the [sys.databases](#) catalog view to learn the recovery model of the **model** database.

```
SELECT name, recovery_model_desc
```

```
FROM sys.databases
    WHERE name = 'model' ;
GO
```

▶ To change the recovery model

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. This example shows how to change the recovery model in the `model` database to `FULL` by using the `SET RECOVERY` option of the [ALTER DATABASE](#) statement.

```
USE master ;
ALTER DATABASE model SET RECOVERY FULL ;
```



Follow Up Recommendations: After You Change the Recovery Model

- **After switching between the full and bulk-logged recovery models**
 - After completing the bulk operations, immediately switch back to full recovery mode.
 - After switching from the bulk-logged recovery model back to the full recovery model, back up the log.

Note

Your backup strategy remains the same: continue performing periodic database, log, and differential backups.

- **After switching from the simple recovery model**
 - Immediately after switching to the full recovery model or bulk-logged recovery model, take a full or differential database backup to start the log chain.

Note

The switch to the full or bulk-logged recovery model takes effect only after the first data backup.

- Schedule regular log backups, and update your restore plan accordingly.

Important

If you do not back up the log frequently enough, the transaction log can expand until it runs out of disk space.

- **After switching to the simple recovery model**

- Discontinue any scheduled jobs for backing up the transaction log.
- Ensure periodic database backups are scheduled. Backing up your database is essential both to protect your data and to truncate the inactive portion of the transaction log.



Related Tasks

- [Create a Full Database Backup \(SQL Server\)](#)
- [Back Up a Transaction Log \(SQL Server\)](#)
- [Create a Job](#)
- [Disable or Enable a Job](#)

Related Content

- [Database Maintenance Plans](#) (in SQL Server 2008 R2 Books Online)



See Also

[Recovery Models \(SQL Server\)](#)

[The Transaction Log \(SQL Server\)](#)

[ALTER DATABASE \(Transact-SQL\)](#)

[sys.databases \(Transact-SQL\)](#)

[Database Recovery Models \(SQL Server\)](#)

Backup Overview

This topic introduces the SQL Server backup component. Backing up your SQL Server database is essential for protecting your data. This discussion covers backup types, and backup restrictions. The topic also introduces SQL Server backup devices and backup media.

In this Topic:

- Components and Concepts
- Backup Compression
- Restrictions on Backup Operations in SQL Server
- Backup Devices and Backup Media
- Related Tasks

Components and Concepts

back up [verb]

Copies the data or log records from a SQL Server database or its transaction log to a backup device, such as a disk, to create a data backup or log backup.

backup [noun]

A copy of SQL Server data that can be used to restore and recover the data after a failure. A backup of SQL Server data is created at the level of a database or one or more of its files or filegroups. Table-level backups cannot be created. In addition to data backups, the full recovery model requires creating backups of the transaction log.

recovery model

A database property that controls transaction log maintenance on a database. Three recovery models exist: simple, full, and bulk-logged. The recovery model of database determines its backup and restore requirements.

restore

A multi-phase process that copies all the data and log pages from a specified SQL Server backup to a specified database, and then rolls forward all the transactions that are logged in the backup by applying logged changes to bring the data forward in time.

Types of Backups

copy-only backup

A special-use backup that is independent of the regular sequence of SQL Server backups.

data backup

A backup of data in a complete database (a database backup), a partial database (a partial backup), or a set of data files or filegroups (a file backup).

database backup

A backup of a database. Full database backups represent the whole database at the time the backup finished. Differential database backups contain only changes made to the database since its most recent full database backup.

differential backup

A data backup that is based on the latest full backup of a complete or partial database or a set of data files or filegroups (the *differential base*) and that contains only the data extents that have changed since the differential base.

A differential partial backup records only the data extents that have changed in the filegroups since the previous partial backup, known as the base for the differential.

full backup

A data backup that contains all the data in a specific database or set of filegroups or

files, and also enough log to allow for recovering that data.

log backup

A backup of transaction logs that includes all log records that were not backed up in a previous log backup. (full recovery model)

file backup

A backup of one or more database files or filegroups.

partial backup

Contains data from only some of the filegroups in a database, including the data in the primary filegroup, every read/write filegroup, and any optionally-specified read-only files.

Backup Media Terms and Definitions

backup device

A disk or tape device to which SQL Server backups are written and from which they can be restored.

backup media

One or more tapes or disk files to which one or more backup have been written.

backup set

The backup content that is added to a media set by a successful backup operation.

media family

Backups created on a single nonmirrored device or a set of mirrored devices in a media set

media set

An ordered collection of backup media, tapes or disk files, to which one or more backup operations have written using a fixed type and number of backup devices.

mirrored media set

Multiple copies (mirrors) of a media set.



Backup Compression

SQL Server 2008 Enterprise and later versions support compressing backups, and SQL Server 2008 and later versions can restore a compressed backup. For more information, see [Backup Compression \(SQL Server\)](#).

Restrictions on Backup Operations in SQL Server

In SQL Server 2005 and later versions, backup can occur while the database is online and being used. However, the following restrictions exist.

Offline Data Cannot Be Backed Up

Any backup operation that implicitly or explicitly references data that is offline fails. Some typical examples include the following:

- You request a full database backup, but one filegroup of the database is offline. Because all filegroups are implicitly included in a full database backup, this operation fails.
To back up this database, you can use a file backup and specify only the filegroups that are online.
- You request a partial backup, but a read/write filegroup is offline. Because all read/write filegroups are required for a partial backup, the operation fails.
- You request a file backup of specific files, but one of the files is not online. The operation fails. To back up the online files, you can omit the offline file from the file list and repeat the operation.

Typically, a log backup succeeds even if one or more data files are unavailable. However, if any file contains bulk-logged changes made under the bulk-logged recovery model, all the files must be online for the backup to succeed.

Concurrency Restrictions During Backup

SQL Server uses an online backup process to allow for a database backup while the database is still being used. During a backup, most operations are possible; for example, INSERT, UPDATE, or DELETE statements are allowed during a backup operation. However, if you try to start a backup operation while a database file is being created or deleted, the backup operation waits until the create or delete operation is finished or the backup times out.

Operations that cannot run during a database backup or transaction log backup include the following:

- File-management operations such as the ALTER DATABASE statement with either the ADD FILE or REMOVE FILE options.
- Shrink database or shrink file operations. This includes auto-shrink operations.
- If you try to create or delete a database file while a backup operation is in progress, the create or delete operation fails.

If a backup operation overlaps with a file-management operation or shrink operation, a conflict occurs. Regardless of which of the conflicting operation began first, the second operation waits for the lock set by the first operation to time out. (The time-out period is controlled by a session time-out setting.) If the lock is released during the time-out period, the second operation continues. If the lock times out, the second operation fails.



Related Tasks

To work with backup devices and backup media

- [Define a Logical Backup Device for a Disk File \(SQL Server\)](#)
- [Define a Logical Backup Device for a Tape Drive \(SQL Server\)](#)
- [Specify a Disk or Tape as a Backup Destination \(SQL Server\)](#)
- [Delete a Backup Device \(SQL Server\)](#)
- [Set the Expiration Date on a Backup \(SQL Server\)](#)
- [View the Contents of a Backup Tape or File \(SQL Server\)](#)
- [View the Data and Log Files In a Backup Set \(SQL Server\)](#)
- [View the Properties and Content of a Backup Device \(SQL Server\)](#)
- [Restore a Backup From a Device \(SQL Server\)](#)

To create a backup



Note

For partial or copy-only backups, you must use the Transact-SQL [BACKUP](#) statement with the PARTIAL or COPY_ONLY option, respectively.

- [Back Up a Database \(SQL Server\)](#)
- [Back Up a Transaction Log \(SQL Server\)](#)
- [Back Up Database Files and Filegroups \(SQL Server\)](#)
- [Create a Differential Database Backup \(SQL Server\)](#)
- [Back Up the Transaction Log When the Database Is Damaged \(SQL Server\)](#)
- [Enable or Disable the Backup CHECKSUM Option \(SQL Server\)](#)
- [Handle CHECKSUM Errors During Backup \(SQL Server\)](#)
- [Use Resource Governor to Limit CPU Usage by Backup Compression \(Transact-SQL\)](#)



See Also

[Back Up and Restore of Databases in SQL Server](#)

[Restore and Recovery Overview \(SQL Server\)](#)

[Maintenance Plans](#)

[Transaction Logs \(SQL Server\)](#)

[Database Recovery Models \(SQL Server\)](#)

Restore and Recovery Overview

SQL Server restore and recovery supports restoring data from backups of a whole database, a data file, or a data page, as follows:

- The database (a *complete database restore*)

The whole database is restored and recovered, and the database is offline for the duration of the restore and recovery operations.

- The data file (a *file restore*)

A data file or a set of files is restored and recovered. During a file restore, the filegroups that contain the files are automatically offline for the duration of the restore. Any attempt to access an offline filegroup causes an error.

- The data page (a *page restore*)

Under the full recovery model or bulk-logged recovery model, you can restore individual databases. Page restores can be performed on any database, regardless of the number of filegroups.

SQL Server backup and restore work across all supported operating systems, whether they are 64-bit or 32-bit systems. For information about the supported operating systems, see [Hardware and Software Requirements for Installing SQL Server](#). For information about support for backups from earlier versions of SQL Server, see the "Compatibility Support" section of [RESTORE \(Transact-SQL\)](#).

In this Topic:

- Overview of Restore Scenarios
- Recovery Models and Supported Restore Operations
- Restore Restrictions Under the Simple Recovery Model
- Restore Under the Bulk-Logged Recovery Model
- Related Tasks
- Related Content

Overview of Restore Scenarios

A *restore scenario* in SQL Server is the process of restoring data from one or more backups and then recovering the database. The supported restore scenarios depend on the recovery model of the database and the edition of SQL Server.

The following table introduces the possible restore scenarios that are supported for different recovery models.

Restore scenario	Under simple recovery model	Under full/bulk-logged recovery models
Complete database restore	This is the basic restore strategy. A complete database restore might involve simply restoring and recovering a full database backup. Alternatively, a complete database restore	This is the basic restore strategy. A complete database restore involve restoring a full database backup and, optionally, a differential backup (if any), followed by restoring all

Restore scenario	Under simple recovery model	Under full/bulk-logged recovery models
	<p>might involve restoring a full database backup followed by restoring and recovering a differential backup.</p> <p>For more information, see Performing a Complete Database Restore (Simple Recovery Model).</p>	<p>subsequent log backups (in sequence). The complete database restore is finished by recovering the last log backup and also restoring it (RESTORE WITH RECOVERY).</p> <p>For more information, see Performing a Complete Database Restore (Full Recovery Model).</p>
File restore *	<p>Restore one or more damaged read-only files, without restoring the entire database. File restore is available only if the database has at least one read-only filegroup.</p>	<p>Restores one or more files, without restoring the entire database. File restore can be performed while the database is offline or, for some editions of SQL Server 2005 and later versions, while the database remains online. During a file restore, the filegroups that contain the files that are being restored are always offline.</p>
Page restore	Not applicable	<p>Restores one or more damaged pages. Page restore can be performed while the database is offline or, for some editions of SQL Server 2005 and later versions, while the database remains online. During a page restore, the pages that are being restored are always offline.</p> <p>An unbroken chain of log backups must be available, up to the current log file, and they must all be applied to bring the page up to date with the current log file.</p>

Restore scenario	Under simple recovery model	Under full/bulk-logged recovery models
		For more information, see Performing Page Restores .
Piecemeal restore *	Restore and recover the database in stages at the filegroup level, starting with the primary and all read/write, secondary filegroups.	Restore and recover the database in stages at the filegroup level, starting with the primary filegroup.

* Online restore is supported only in SQL Server 2005 Enterprise Edition and later versions.

Regardless of how data is restored, before a database can be recovered, the SQL Server Database Engine guarantees that the whole database is logically consistent. For example, if you restore a file, you cannot recover it and bring it online until it has been rolled far enough forward to be consistent with the database.

Advantages of a File or Page Restore

Restoring and recovering files or pages, instead of the whole database, provides the following advantages:

- Restoring less data reduces the time required to copy and recover it.
- On SQL Server 2005 Enterprise Edition and later versions, restoring files or pages might allow other data in the database to remain online during the restore operation.

Recovery Models and Supported Restore Operations

The restore operations that are available for a database depend on its recovery model. The following table summarizes whether and to what extent each of the recovery models supports a given restore scenario.

Restore operation	Full recovery model	Bulk-logged recovery model	Simple recovery model
Data recovery	Complete recovery (if the log is available).	Some data-loss exposure.	Any data since last full or differential backup is lost.

Restore operation	Full recovery model	Bulk-logged recovery model	Simple recovery model
Point-in-time restore	Any time covered by the log backups.	Disallowed if the log backup contains any bulk-logged changes.	Not supported.
File restore *	Full support.	Sometimes.**	Available only for read-only secondary files.
Page restore *	Full support.	Sometimes.**	None.
Piecemeal (filegroup-level) restore *	Full support.	Sometimes.**	Available only for read-only secondary files.

* Available only in the SQL Server 2005 Enterprise Edition and later versions.

** For the required conditions, see Restore Restrictions Under the Simple Recovery Model, later in this topic.

Important

Regardless of the recovery model of a database, a SQL Server backup cannot be restored by a version of SQL Server that is older than the version that created the backup. Thus, for example, a backup created on SQL Server 2012 cannot be restored by SQL Server 2008.

Restore Scenarios Under the Simple Recovery Model

The simple recovery model imposes the following restrictions on restore operations:

- File restore and piecemeal restore are available only for read-only secondary filegroups. For information about these restore scenarios, see [Restoring File Backups \(Simple Recovery Model\)](#) and [Performing Piecemeal Restores](#).
- Page restore is not allowed.
- Point-in-time restore is not allowed.

If any of these restrictions are inappropriate for your recovery needs, we recommend that you consider using the full recovery model. For more information, see [Backup Overview \(SQL Server\)](#).

Important

Regardless of the recovery model of a database, a SQL Server backup cannot be restored by a version of SQL Server that is older than the version that created the

backup. Thus, for example, a backup created on SQL Server 2012 cannot be restored by SQL Server 2008.

Restore Under the Bulk-Logged Recovery Model

This section discusses restore considerations that are unique to bulk-logged recovery model, which is intended exclusively as a supplement to the full recovery model.

Note

For an introduction to the bulk-logged recovery model, see [Transaction Logs \(SQL Server\)](#).

Generally, the bulk-logged recovery model is similar to the full recovery model, and the information described for the full recovery model also applies to both. However, point-in-time recovery and online restore are affected by the bulk-logged recovery model.

Restrictions for Point-in-time Recovery

If a log backup taken under the bulk-logged recovery model contains bulk-logged changes, point-in-time recovery is not allowed. Trying to perform point-in-time recovery on a log backup that contains bulk changes will cause the restore operation to fail.

Restrictions for Online Restore

An online restore sequence works only if the following conditions are met:

- All required log backups must have been taken before the restore sequence starts.
- Bulk changes must be backed before starting the online restore sequence.
- If bulk changes exist in the database, all files must be either online or defunct. (This means that it is no longer part of the database.)

If these conditions are not met, the online restore sequence fails.

Note

We recommend switching to the full recovery model before starting an online restore. For more information, see [Database Recovery Models \(SQL Server\)](#).

For information about how to perform an online restore, see [Performing Online Restores](#).

Related Tasks

Restoring Data Backups

Using SQL Server Management Studio

- [How to: Restore a Database Backup \(SQL Server Management Studio\)](#)
- [How to: Create a New Database From An Existing Database Backup \(SQL Server Management Studio\)](#)
- [How to: Restore a Differential Backup \(SQL Server Management Studio\)](#)
- [How to: Restore Files and Filegroups \(SQL Server Management Studio\)](#)

Using Transact-SQL

- [How to: Restore a Database Backup Under the Simple Recovery Model \(Transact-SQL\)](#)
- [How to: Restore a Database to the Point of Failure Under the Full Recovery Model \(Transact-SQL\)](#)
- [How to: Restore Files and Filegroups over Existing Files \(Transact-SQL\)](#)
- [How to: Restore Files to a New Location \(Transact-SQL\)](#)
- [How to: Restore the master Database \(Transact-SQL\)](#)

Restoring Transaction Logs (Full Recovery Model)

Using SQL Server Management Studio

- [How to: Restore a Database to a Marked Transaction \(SQL Server Management Studio\)](#)
- [How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)
- [Restore a Database to a Point In Time \(SQL Server\)](#)

Using Transact-SQL

- [Plan and Perform Restore Sequences \(Full Recovery Model\)](#)
- [Restore a Database to a Point In Time \(SQL Server\)](#)

Additional Restore Tasks

Using Transact-SQL

- [How to: Restart an Interrupted Restore Operation \(Transact-SQL\)](#)
- [How to: Recover a Database from a Backup Without Restoring Data \(Transact-SQL\)](#)

Related Content

None.

See Also

[Overview of Backup in SQL Server](#)

Plan and Perform Restore Sequences (Full Recovery Model)

This topic explains how to plan and perform a restore sequence for a SQL Server databases that ordinarily uses the full recovery model. A *restore sequence* is a sequence of one or more [RESTORE](#) statements. Typically, a restore sequences initializes the contents of the database, files, and/or pages being restored (the data-copy phase), rolls forward logged transactions (the redo phase), and rolls back uncommitted transactions (the undo phase).

In simple cases, a restore sequence requires only a full database backup, a differential database backup, and the subsequent log backups. In these cases, constructing a correct restore sequence is easy. For example, to restore a whole database to the point of a failure, start by backing up the active transaction log (the *tail* of the log). Then, restore the most recent full database backup, the most recent differential backup (if any), and all subsequent log backups in the order in which they were taken.

In more complex cases, constructing a correct restore sequence can be a complex process. For example, a restore sequence might require multiple file backups or restoring data to a specific point in time. In very complex cases, you might even have to traverse a forked recovery path that spans one or more recovery forks.



Note

A *recovery path* is the sequence of data and log backups that have brought a database to a particular point in time (known as a recovery point). A recovery path is a specific set of transformations that have evolved the database over time, yet have maintained the consistency of the database. A recovery path describes a range of LSNs from a start point (LSN,GUID) to an end point (LSN,GUID). The range of LSNs in a recovery path can traverse one or more recovery branches from start to end.

To Plan a Restore Sequence

Before you start a restore sequence, follow these steps:

1. Create a tail-log backup of the database, if you can. For more information, see [Tail-Log Backups](#).
2. Determine the target recovery point.

The target recovery point can be any point in time or mark within a transaction log backup. For more information, see [Restore a Database to a Point in Time](#) or [Ensuring Recoverability of Related Databases \(Using Marked Transactions\)](#).

3. Determine the type of restore you want to perform. For more information, see [Overview of Restore and Recovery in SQL Server](#).
4. Identify which backups you require and make sure that the necessary media sets and backup devices are available. For more information, see [Backup Devices \(SQL Server\)](#) and [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

To Perform a Restore Sequence

To perform a restore sequence, follow these steps:

1. To start the sequence, restore a one or more data backups, such as: a database backup, a partial backup, one or more file backups.
2. Optionally, restore the latest differential backups that are based on these full backups.

For each full backup that you plan to restore, determine whether it is the base for any differential backups. If so, restore most recent differential backup, if you can. For more information, see [Use Differential Backups](#).

3. Roll forward the database by restoring log backups in sequence, finishing with the backup that contains the recovery point. Whether you have to apply all the log backups depends on what log backup contains the target recovery point, as follows:
 - If the recovery point is the point of a failure, you must restore every log backup that was created since the last data (full or differential) backup you restored. For more information, see [Applying Transaction Log Backups](#).
 - For a point-in-time restore, you might not require the most recent log backups. For more information, see [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).

Restarting a Restore Sequence

If you encounter a problem with the outcome of a restore sequence, you can quit it and restart the restore sequence over from the start. For example, if you accidentally restore too many log backups and overshoot the intended recovery point, you must restart the restore sequence up to log backup that contains the target recovery point.

See Also

[Overview of Backup in SQL Server](#)

[Overview of Restore and Recovery in SQL Server](#)

[Complete Database Restores \(Full Recovery Model\)](#)

[Online Restore \(SQL Server\)](#)

[File Restores \(Full Recovery Model\)](#)

[Page Restores](#)

[Piecemeal Restores](#)

Restart an Interrupted Restore Operation (Transact-SQL)

This topic explains how to restart an interrupted restore operation.

Procedures

▶ To restart an interrupted restore operation

1. Execute the interrupted RESTORE statement again, specifying:
 - The same clauses used in the original RESTORE statement.
 - The RESTART clause.

Example

Description

This example restarts an interrupted restore operation.

Code

```
-- Restore a full database backup of the AdventureWorks database.
RESTORE DATABASE AdventureWorks
    FROM DISK = 'C:\AdventureWorks.bck'
GO
-- The restore operation halted prematurely.
-- Repeat the original RESTORE statement specifying WITH RESTART.
RESTORE DATABASE AdventureWorks
    FROM DISK = 'C:\AdventureWorks.bck'
    WITH RESTART
GO
```

See Also

[RESTORE \(Transact-SQL\)](#)

[Performing a Complete Database Restore \(Simple Recovery Model\)](#)

[RESTORE](#)

Backup Compression

This topic describes the compression of SQL Server backups, including restrictions, performance trade-off of compressing backups, the configuration of backup compression, and the compression ratio.

Note

For information which editions of SQL Server 2012 support backup compression, see [Features Supported by the Editions of SQL Server 2012](#). Every edition of SQL Server 2008 and later can restore a compressed backup.

In this Topic:

- Benefits
- Restrictions
- Performance Impact of Compressing Backups
- Calculate the Compression Ratio of a Compressed Backup
- Allocation of Space for the Backup File
- Related Tasks

Benefits

- Because a compressed backup is smaller than an uncompressed backup of the same data, compressing a backup typically requires less device I/O and therefore usually increases backup speed significantly.

For more information, see Performance Impact of Compressing Backups, later in this topic.

Restrictions

The following restrictions apply to compressed backups:

- Compressed and uncompressed backups cannot co-exist in a media set.
- Previous versions of SQL Server cannot read compressed backups.
- NTbackups cannot share a tape with compressed SQL Server backups.

Performance Impact of Compressing Backups

By default, compression significantly increases CPU usage, and the additional CPU consumed by the compression process might adversely impact concurrent operations. Therefore, you might want to create low-priority compressed backups in a session whose CPU usage is limited by [Resource Governor](#). For more information, see [How to: Use Resource Governor to Limit CPU Usage by Backup Compression \(Transact-SQL\)](#).

To obtain a good picture of your backup I/O performance, you can isolate the backup I/O to or from devices by evaluating the following sorts of performance counters:

- Windows I/O performance counters, such as the physical-disk counters
- The **Device Throughput Bytes/sec** counter of the [SQLServer:Backup Device](#) object
- The **Backup/Restore Throughput/sec** counter of the [SQLServer:Databases](#) object

For information about Windows counters, see Windows help. For information about how to work with SQL Server counters, see [Using SQL Server Objects](#).

Calculate the Compression Ratio of a Compressed Backup

To calculate the compression ratio of a backup, use the values for the backup in the **backup_size** and **compressed_backup_size** columns of the [backupset](#) history table, as follows:

backup_size:compressed_backup_size

For example, a 3:1 compression ratio indicates that you are saving about 66% on disk space. To query on these columns, you can use the following Transact-SQL statement:

```
SELECT backup_size/compressed_backup_size FROM msdb..backupset;
```

The compression ratio of a compressed backup depends on the data that has been compressed. A variety of factors can impact the compression ratio obtained. Major factors include:

- The type of data.
Character data compresses more than other types of data.
- The consistency of the data among rows on a page.
Typically, if a page contains several rows in which a field contains the same value, significant compression might occur for that value. In contrast, for a database that contains random data or that contains only one large row per page, a compressed backup would be almost as large as an uncompressed backup.
- Whether the data is encrypted.
Encrypted data compresses significantly less than equivalent unencrypted data. If transparent data encryption is used to encrypt an entire database, compressing backups might not reduce their size by much, if at all.
- Whether the database is compressed.
If the database is compressed, compressing backups might not reduce their size by much, if at all.

Allocation of Space for the Backup File

For compressed backups, the size of the final backup file depends on how compressible the data is, and this is unknown before the backup operation finishes. Therefore, by default, when backing up a database using compression, the Database Engine uses a pre-allocation algorithm for the backup file. This algorithm pre-allocates a predefined percentage of the size of the database for the backup file. If more space is needed during the backup operation, the Database Engine grows the file. If the final size is less than the allocated space, at the end of the backup operation, the Database Engine shrinks the file to the actual final size of the backup.

To allow the backup file to grow only as needed to reach its final size, use trace flag 3042. Trace flag 3042 causes the backup operation to bypass the default backup compression pre-allocation algorithm. This trace flag is useful if you need to save on space by allocating only the actual size required for the compressed backup. However, using this trace flag might cause a slight performance penalty (a possible increase in the duration of the backup operation).

Related Tasks

- [Configure Backup Compression](#)
- [View or Configure the backup compression default Option \(SQL Server Management Studio\)](#)
- [Use Resource Governor to Limit CPU Usage by Backup Compression \(Transact-SQL\)](#)
- [DBCC TRACEON \(Transact-SQL\)](#)
- [DBCC TRACEOFF \(Transact-SQL\)](#)

See Also

[Backup Overview \(SQL Server\)](#)

[Trace Flages \(Transact-SQL\)](#)

Configure Backup Compression

At installation, backup compression is off by default. The default behavior for backup compression is defined by the **backup compression default** Option server-level configuration option. However, you can override the server-level default when creating a single backup or scheduling a series of routine backups. To change the server-level default, see [View or Configure the backup compression default Server Configuration Option](#).

Override the Backup Compression Default

You can change the backup compression behavior for an individual backup, backup job, or log shipping configuration.

- **Transact-SQL**

To override the server backup-compression default when creating a backup, use either `WITH NO_COMPRESSION` or `WITH COMPRESSION` in your [BACKUP](#) statement. For a log shipping configuration, you can control the backup compression behavior of log backups by using [sp_add_log_shipping_primary_database](#) or [sp_change_log_shipping_primary_database](#).

- **SQL Server Management Studio**

For information about how to view or configure the backup compression default option for an instance of SQL Server, see [View and Configure the backup compression default Option \(SQL Server Management Studio\)](#).

You can override the server backup-compression default when creating a backup by specifying **Compress backup** or **Do not compress backup** in any of the following dialog boxes:

- **Back Up Database (Options Page)**
When backing up a database, you can control backup compression for an individual database, file, or log backup.
- [Maintenance Plan Wizard](#)
The Maintenance Plan Wizard enables you to control backup compression for each set full or differential database backups or log backups that you schedule.
- **Integration Services (SSIS) [Back Up Database task](#)**
You can control the backup compression behavior when creating a package for backing up a single database or multiple databases.
- [Log Shipping Transaction Log Backup Settings](#)
You can control the backup compression behavior of log backups.

See Also

[Backup Compression \(SQL Server\)](#)

Use Resource Governor to Limit CPU Usage by Backup Compression (Transact-SQL)

By default, backing up using compression significantly increases CPU usage, and the additional CPU consumed by the compression process can adversely impact concurrent operations. Therefore, you might want to create a low-priority compressed backup in a session whose CPU usage is limited by [Resource Governor](#) when CPU contention occurs. This topic presents a scenario that classifies the sessions of a particular SQL Server user by mapping them to a Resource Governor workload group that limits CPU usage in such cases.

Important

In a given Resource Governor scenario, session classification might be based on a user name, an application name, or anything else that can differentiate a connection. For more information, see [Resource Governor Classifier Function](#) and [Resource Governor Workload Group](#).

This topic contains the following set of scenarios, which are presented in sequence:

1. Setting Up a Login and User for Low-Priority Operations
2. Configuring Resource Governor to Limit CPU Usage
3. Verifying the Classification of the Current Session (Transact-SQL)
4. Compressing Backups Using a Session with Limited CPU

Setting Up a Login and User for Low-Priority Operations

The scenario in this topic requires a low-priority SQL Server login and user. The user name will be used to classify sessions running in the login and route them to a Resource Governor workload group that limits CPU usage.

The following procedure describes the steps for setting up a login and user for this purpose, followed by a Transact-SQL example, "Example A: Setting Up a Login and User (Transact-SQL)."

To set up a login and database user for classifying sessions

1. Create a SQL Server login for creating low-priority compressed backups.

To create a login

- [How To: Create A SQL Server Login](#)
- [CREATE LOGIN \(Transact-SQL\)](#)

2. Optionally, grant VIEW SERVER STATE to this login.

- [GRANT System Object Permissions \(Transact-SQL\)](#)

For more information, see [GRANT Database Principal Permissions \(Transact-SQL\)](#).

3. Create a SQL Server user for this login.

To create a user

- [How To: Create a Database User](#)
- [CREATE USER \(Transact-SQL\)](#)

4. To enable sessions of this login and user to back up a given database, add the user to the db_backupoperator database role of that database. Do this for each database that this user will back up. Optionally, add the user to other fixed database roles.

To add a user to a fixed database role

- [sp_addrolemember \(Transact-SQL\)](#)

For more information, see [GRANT Database Principal Permissions \(Transact-SQL\)](#).

Example A: Setting Up a Login and User (Transact-SQL)

The following example is relevant only if you choose to create a new SQL Server login and user for low-priority backups. Alternatively, you can use an existing login and user, if an appropriate one exists.

Important

The following example uses a sample login and user name, *domain_name*\MAX_CPU. Replace these with the names of the SQL Server login and user that you plan to use when creating your low-priority compressed backups.

This example creates a login for the *domain_name*\MAX_CPU Windows account and then grants VIEW SERVER STATE permission to the login. This permission enables you to verify the Resource Governor classification of sessions of the login. The example then creates a user for *domain_name*\MAX_CPU and adds it to the db_backupoperator fixed database role for the AdventureWorks2012 sample database. This user name will be used by the Resource Governor classifier function.

```
-- Create a SQL Server login for low-priority operations
USE master;
CREATE LOGIN [domain_name\MAX_CPU] FROM WINDOWS;
GRANT VIEW SERVER STATE TO [domain_name\MAX_CPU];
GO

-- Create a SQL Server user in AdventureWorks2012 for this login
USE AdventureWorks2012;
CREATE USER [domain_name\MAX_CPU] FOR LOGIN [domain_name\MAX_CPU];
EXEC sp_addrolemember 'db_backupoperator', 'domain_name\MAX_CPU';
GO
```

Configuring Resource Governor to Limit CPU Usage

Note

Ensure that Resource Governor is enabled. For more information, see [How to: Enable or Disable Resource Governor](#).

In this Resource Governor scenario, configuration comprises the following basic steps:

1. Create and configure a Resource Governor resource pool that limits the maximum average CPU bandwidth that will be given to requests in the resource pool when CPU contention occurs.
2. Create and configure a Resource Governor workload group that uses this pool.

3. Create a *classifier function*, which is a user-defined function (UDF) whose return values are used by Resource Governor for classifying sessions so that they are routed to the appropriate workload group.
4. Register the classifier function with Resource Governor.
5. Apply the changes to the Resource Governor in-memory configuration.



Note

For information about Resource Governor resource pools, workload groups, and classification, see [Resource Governor](#).

The Transact-SQL statements for these steps are described in the procedure, "To configure Resource Governor for limiting CPU usage," which is followed by a Transact-SQL example of the procedure.

To configure Resource Governor (SQL Server Management Studio)

- [How to: Configure Resource Governor Using a Template](#)
- [How to: Create a Resource Pool \(SQL Server Management Studio\)](#)
- [How to: Create a Workload Group \(SQL Server Management Studio\)](#)

To configure Resource Governor for limiting CPU usage (Transact-SQL)

1. Issue a [CREATE RESOURCE POOL](#) statement to create a resource pool. The example for this procedure uses the following syntax:

```
CREATE RESOURCE POOL pool_name WITH ( MAX_CPU_PERCENT = value );
```

Value is an integer from 1 to 100 that indicates the percentage of maximum average CPU bandwidth. The appropriate value depends on your environment. For the purpose of illustration, the example in this topic uses 20% percent (MAX_CPU_PERCENT = 20.)

2. Issue a [CREATE WORKLOAD GROUP](#) statement to create a workload group for low-priority operations whose CPU usage you want to govern. The example for this procedure uses the following syntax:

```
CREATE WORKLOAD GROUP group_name USING pool_name;
```

3. Issue a [CREATE FUNCTION](#) statement to create a classifier function that maps the workload group created in the preceding step to the user of the low-priority login. The example for this procedure uses the following syntax:

```
CREATE FUNCTION [schema_name.]function_name() RETURNS sysname
WITH SCHEMABINDING
```

```
AS
```

```
BEGIN
```

```
    DECLARE @workload_group_name AS sysname
```

```
    IF (SUSER_NAME() = 'user_of_low_priority_login')
```

```
        SET @workload_group_name = 'workload_group_name'
```

```
RETURN @workload_group_name
```

```
END
```

For information about the components of this CREATE FUNCTION statement, see:

- [DECLARE @local_variable \(Transact-SQL\)](#)
- [SUSER_SNAME \(Transact-SQL\)](#)



Important

SUSER_NAME is just one of several system functions that can be used in a classifier function. For more information, see [Create and Test a Classifier User-Defined Function](#).

- [SET @local_variable \(Transact-SQL\)](#).
4. Issue an [ALTER RESOURCE GOVERNOR](#) statement to register the classifier function with Resource Governor. The example for this procedure uses the following syntax:
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION =
schema_name.function_name);
 5. Issue a second ALTER RESOURCE GOVERNOR statement to apply the changes to the Resource Governor in-memory configuration, as follows:

```
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

Example B: Configuring Resource Governor (Transact-SQL)

The following example performs the following steps within a single transaction:

1. Creates the `pMAX_CPU_PERCENT_20` resource pool.
2. Creates the `gMAX_CPU_PERCENT_20` workload group.
3. Creates the `rgclassifier_MAX_CPU()` classifier function, which uses the user name created in the preceding example.
4. Registers the classifier function with Resource Governor.

After committing the transaction, the example applies the configuration changes requested in the ALTER WORKLOAD GROUP or ALTER RESOURCE POOL statements.



Important

The following example uses the user name of the sample SQL Server user created in "Example A: Setting Up a Login and User (Transact-SQL)," `domain_name\MAX_CPU`. Replace this with the name of the user of the login that you plan to use for creating low-priority compressed backups.

```
-- Configure Resource Governor.
```

```
BEGIN TRAN
```

```
USE master;
```

```
-- Create a resource pool that sets the MAX_CPU_PERCENT to 20%.
```

```
CREATE RESOURCE POOL pMAX_CPU_PERCENT_20
```

```

WITH
    (MAX_CPU_PERCENT = 20);
GO
-- Create a workload group to use this pool.
CREATE WORKLOAD GROUP gMAX_CPU_PERCENT_20
USING pMAX_CPU_PERCENT_20;
GO
-- Create a classification function.
-- Note that any request that does not get classified goes into
-- the 'Default' group.
CREATE FUNCTION dbo.rgclassifier_MAX_CPU() RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @workload_group_name AS sysname
        IF (SUSER_NAME() = 'domain_name\MAX_CPU')
            SET @workload_group_name = 'gMAX_CPU_PERCENT_20'
    RETURN @workload_group_name
END;
GO

-- Register the classifier function with Resource Governor.
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION=
dbo.rgclassifier_MAX_CPU);
COMMIT TRAN;
GO
-- Start Resource Governor
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO

```

Verifying the Classification of the Current Session (Transact-SQL)

Optionally, log in as the user that you specified in your classifier function, and verify the session classification by issuing the following [SELECT](#) statement in Object Explorer:

```

USE master;
SELECT sess.session_id, sess.login_name, sess.group_id, grps.name
FROM sys.dm_exec_sessions AS sess
JOIN sys.dm_resource_governor_workload_groups AS grps
    ON sess.group_id = grps.group_id
WHERE session_id > 50;
GO

```

In the results pane, the **name** column should list one or more sessions for the workload-group name that you specified in your classifier function.



Note

For information about the dynamic management views called by this SELECT statement, see [sys.dm_exec_sessions \(Transact-SQL\)](#) and [sys.dm_resource_governor_workload_groups \(Transact-SQL\)](#).

Compressing Backups Using a Session with Limited CPU

To create a compressed backup in a session with a limited maximum CPU, log in as the user specified in your classifier function. In your backup command, either specify **WITH COMPRESSION** (Transact-SQL) or select **Compress backup** (SQL Server Management Studio). To create a compressed database backup, see [Create a Full Database Backup](#).

Example C: Creating a Compressed Backup (Transact-SQL)

The following [BACKUP](#) example creates a compressed full backup of the AdventureWorks2012 database in a newly formatted backup file,

```
Z:\SQLServerBackups\AdvWorksData.bak.
```

```
--Run backup statement in the gBackup session.
```

```
BACKUP DATABASE AdventureWorks2012 TO
```

```
DISK='Z:\SQLServerBackups\AdvWorksData.bak'
```

```
WITH
```

```
    FORMAT,
```

```
    MEDIADESCRIPTION='AdventureWorks2012 Compressed Data Backups'
```

```
    DESCRIPTION='First database backup on AdventureWorks2012 Compressed
Data Backups media set'
```

```
    COMPRESSION;
```

```
GO
```

See Also

Full Database Backups

A full database backup backs up the whole database. This includes part of the transaction log so that the full database can be recovered after a full database backup is restored. Full database backups represent the database at the time the backup finished.

Tip

As a database increases in size full database backups take more time to finish and require more storage space. Therefore, for a large database, you might want to supplement a full database backup with a series of *differential database backups*. For more information, see [Differential Backups \(SQL Server\)](#).

noteDXDOC112778PADS **Security Note**

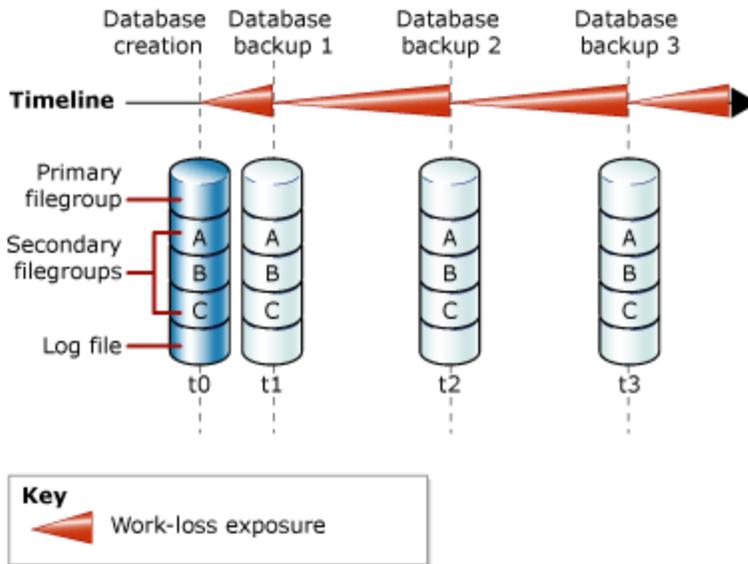
TRUSTWORTHY is set to OFF on a database backup. For information about how to set TRUSTWORTHY to ON, see [ALTER DATABASE SET Options \(Transact-SQL\)](#).

In This Topic:

- Database Backups Under the Simple Recovery Model
- Database Backups Under the Full Recovery Model
- Use a Full Database Backup to Restore the Database
- Related Tasks

Database Backups Under the Simple Recovery Model

Under the simple recovery model, after each backup, the database is exposed to potential work loss if a disaster were to occur. The work-loss exposure increases with each update until the next backup, when the work-loss exposure returns to zero and a new cycle of work-loss exposure starts. Work-loss exposure increases over time between backups. The following illustration shows the work-loss exposure for a backup strategy that uses only full database backups.



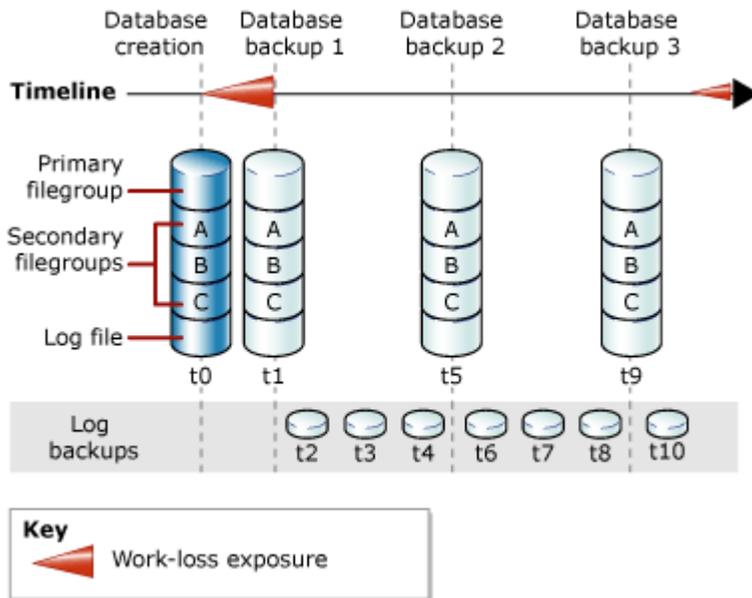
Example (Transact-SQL)

The following example shows how to create a full database backup by using WITH FORMAT to overwrite any existing backups and create a new media set.

```
-- Back up the AdventureWorks2012 database to new media set.
BACKUP DATABASE AdventureWorks2012
    TO DISK = 'Z:\SQLServerBackups\AdventureWorksSimpleRM.bak'
    WITH FORMAT;
GO
```

Database Backups Under the Full Recovery Model

For databases that use full and bulk-logged recovery, database backups are necessary but not sufficient. Transaction log backups are also required. The following illustration shows the least complex backup strategy that is possible under the full recovery model.



For information about how to create log backups, see [Create Transaction Log Backups \(SQL Server\)](#).

Example (Transact-SQL)

The following example shows how to create a full database backup by using WITH FORMAT to overwrite any existing backups and create a new media set. Then, the example backs up the transaction log. In a real-life situation, you would have to perform a series of regular log backups. For this example, the sample database is set to use the full recovery model.

```
USE master;
ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;
GO
-- Back up the AdventureWorks2012 database to new media set (backup set 1).
BACKUP DATABASE AdventureWorks2012
    TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012FullRM.bak'
    WITH FORMAT;
GO
--Create a routine log backup (backup set 2).
BACKUP LOG AdventureWorks2012 TO DISK =
'Z:\SQLServerBackups\AdventureWorks2012FullRM.bak';
GO
```

Use a Full Database Backup to Restore the Database

You can re-create a whole database in one step by restoring the database from a full database backup to any location. Enough of the transaction log is included in the backup to let you recover the database to the time when the backup finished. The restored database matches the state of the original database when the database backup finished, minus any uncommitted transactions. Under the full recovery model, you should then restore all subsequent transaction log backups. When the database is recovered, uncommitted transactions are rolled back.

For more information, see [Complete Database Restore \(Simple Recovery Model\)](#) or [Complete Database Restore \(Full Recovery Model\)](#).

Related Tasks

To create a full database backup

- [Create a Full Database Backup \(SQL Server\)](#)

-

`M:Microsoft.SqlServer.Management.Smo.Backup.SqlBackup(Microsoft.SqlServer.Management.Smo.Server)` (SMO)

To schedule backup jobs

[Use the Maintenance Plan Wizard](#)

See Also

[Back Up and Restore of SQL Server Databases](#)

[Backup Overview \(SQL Server\)](#)

[Backing Up and Restoring an Analysis Services Database](#)

Create a Full Database Backup

This topic describes how to create a full database backup in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Recommendations
 - Security
- **To create a full database backup, using:**
 - SQL Server Management Studio

Transact-SQL

- Related Tasks

Before You Begin

Limitations and Restrictions

- The BACKUP statement is not allowed in an explicit or implicit transaction.
- Backups that are created by more recent version of SQL Server cannot be restored in earlier versions of SQL Server.
- For more information, see [Backup Overview \(SQL Server\)](#).

Recommendations

- As a database increases in size full database backups take more time to finish and require more storage space. Therefore, for a large database, you might want to supplement a full database backup with a series of *differential database backups*. For more information, see [Differential Backups \(SQL Server\)](#).
- You can estimate the size of a full database backup by using the [sp_spaceused](#) system stored procedure.
- By default, every successful backup operation adds an entry in the SQL Server error log and in the system event log. If back up the log very frequently, these success messages accumulate quickly, resulting in huge error logs that can make finding other messages difficult. In such cases you can suppress these log entries by using trace flag 3226 if none of your scripts depend on those entries. For more information, see [Trace Flags \(Transact-SQL\)](#).

Security

TRUSTWORTHY is set to OFF on a database backup. For information about how to set TRUSTWORTHY to ON, see [ALTER DATABASE SET Options \(Transact-SQL\)](#).

Beginning with SQL Server 2012 the **PASSWORD** and **MEDIAPASSWORD** options are discontinued for creating backups. You can still restore backups created with passwords.

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio



Note

When you specify a back up task by using SQL Server Management Studio, you can generate the corresponding Transact-SQL [BACKUP](#) script by clicking the **Script** button and selecting a script destination.

▶ To back up a database

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Database** list box, verify the database name. You can optionally select a different database from the list.
5. You can perform a database backup for any recovery model (**FULL**, **BULK_LOGGED**, or **SIMPLE**).
6. In the **Backup type** list box, select **Full**.

Note that after creating a full database backup, you can create a differential database backup; for more information, see [SQL Server Management Studio Tutorial](#).

7. Optionally, you can select **Copy Only Backup** to create a copy-only backup. A *copy-only backup* is a SQL Server backup that is independent of the sequence of conventional SQL Server backups. For more information, see [Copy-Only Backups](#).



Note

When the **Differential** option is selected, you cannot create a copy-only backup.

8. For **Backup component**, click **Database**.
9. Either accept the default backup set name suggested in the **Name** text box, or enter a different name for the backup set.
10. Optionally, in the **Description** text box, enter a description of the backup set.
11. Specify when the backup set will expire and can be overwritten without explicitly

skipping verification of the expiration data:

- To have the backup set expire after a specific number of days, click **After** (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.

The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (Database Settings Page). To access this, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.

- To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.

For more information about backup expiration dates, see [BACKUP \(Transact-SQL\)](#).

12. Choose the type of backup destination by clicking **Disk** or **Tape**. To select the paths of up to 64 disk or tape drives containing a single media set, click **Add**. The selected paths are displayed in the **Backup to** list box.

To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.

13. To view or select the advanced options, click **Options** in the **Select a page** pane.

14. Select an **Overwrite Media** option, by clicking one of the following:

- **Back up to the existing media set**

For this option, click either **Append to the existing backup set** or **Overwrite all existing backup sets**. For more information, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

Optionally, select **Check media set name and backup set expiration** to cause the backup operation to verify the date and time at which the media set and backup set expire.

Optionally, enter a name in the **Media set name** text box. If no name is specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see whether the actual name matches the name you enter here.

- **Back up to a new media set, and erase all existing backup sets**

For this option, enter a name in the **New media set name** text box, and, optionally, describe the media set in the **New media set description** text box.

15. In the **Reliability** section, optionally check:

- **Verify backup when finished**.
- **Perform checksum before writing to media**, and, optionally, **Continue on checksum error**. For information on checksums, see [Detecting and Coping](#)

[with Media Errors.](#)

16. If you are backing up to a tape drive (as specified in the **Destination** section of the **General** page), the **Unload the tape after backup** option is active. Clicking this option activates the **Rewind the tape before unloading** option.



Note

The options in the **Transaction log** section are inactive unless you are backing up a transaction log (as specified in the **Backup type** section of the **General** page).

17. SQL Server 2008 Enterprise and later supports [backup compression](#). By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by checking **Compress backup**, and you can prevent compression by checking **Do not compress backup**.

To view or change the current backup compression default

- [How to: View and Configure the backup compression default Option \(SQL Server Management Studio\)](#)



Note

Alternatively, you can use the Maintenance Plan Wizard to create database backups.



Using Transact-SQL

▶ To create a full database backup

1. Execute the BACKUP DATABASE statement to create the full database backup, specifying:
 - The name of the database to back up.
 - The backup device where the full database backup is written.

The basic Transact-SQL syntax for a full database backup is:

```
BACKUP DATABASE database
    TO backup_device [ ,...n ]
    [ WITH with_options [ ,...o ] ] ;
```

Option	Description
database	Is the database that is to be backed up.
backup_device [,...n]	Specifies a list of from 1 to 64 backup

	<p>devices to use for the backup operation. You can specify a physical backup device, or you can specify a corresponding logical backup device, if already defined. To specify a physical backup device, use the DISK or TAPE option:</p> <p>{ DISK TAPE } = physical_backup_device_name</p> <p>For more information, see Backup Devices.</p>
WITH with_options [,...o]	<p>Optionally, specifies one or more additional options, o. For information about some of the basic with options, see step 2.</p>

- Optionally, specify one or more WITH options. A few basic WITH options are described here. For information about all the WITH options, see [BACKUP \(Transact-SQL\)](#).

- Basic backup set WITH options:

{ COMPRESSION | NO_COMPRESSION }

In SQL Server 2008 Enterprise and later only, specifies whether [backup compression](#) is performed on this backup, overriding the server-level default.

DESCRIPTION = { 'text' | @text_variable }

Specifies the free-form text that describes the backup set. The string can have a maximum of 255 characters.

NAME = { backup_set_name | @backup_set_name_var }

Specifies the name of the backup set. Names can have a maximum of 128 characters. If NAME is not specified, it is blank.

- Basic backup set WITH options:

By default, BACKUP appends the backup to an existing media set, preserving existing backup sets. To explicitly specify this, use the NOINIT option. For information about appending to existing backup sets, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

Alternatively, to format the backup media, use the FORMAT option:

**FORMAT [, MEDIUMNAME= { media_name | @media_name_variable }] [,
MEDIADescription = { text | @text_variable }]**

Use the FORMAT clause when you are using media for the first time or you

want to overwrite all existing data. Optionally, assign the new media a media name and description.

 **Important**

Use extreme caution when you are using the `FORMAT` clause of the `BACKUP` statement because this destroys any backups that were previously stored on the backup media.

Examples (Transact-SQL)

A. Backing up to a disk device

The following example backs up the complete `AdventureWorks2012` database to disk, by using `FORMAT` to create a new media set.

```
USE AdventureWorks2012;
GO
BACKUP DATABASE AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.Bak'
WITH FORMAT,
    MEDIANAME = 'Z_SQLServerBackups',
    NAME = 'Full Backup of AdventureWorks2012';
GO
```

B. Backing up to a tape device

The following example backs up the complete `AdventureWorks2012` database to tape, appending the backup to the previous backups.

```
USE AdventureWorks2012;
GO
BACKUP DATABASE AdventureWorks2012
TO TAPE = '\\.\Tape0'
WITH NOINIT,
    NAME = 'Full Backup of AdventureWorks2012';
GO
```

C. Backing up to a logical tape device

The following example creates a logical backup device for a tape drive. The example then backs up the complete `AdventureWorks2012` database to that device.

```
-- Create a logical backup device,
-- AdventureWorks2012_Bak_Tape, for tape device \\.\tape0.
```

```
USE master;
GO
EXEC sp_addumpdevice 'tape', 'AdventureWorks2012_Bak_Tape',
'\\.\tape0';
USE AdventureWorks2012;
GO
BACKUP DATABASE AdventureWorks2012
    TO AdventureWorks2012_Bak_Tape
    WITH FORMAT,
        MEDIANAME = 'AdventureWorks2012_Bak_Tape',
        MEDIADESCRIPTION = '\\.\tape0',
        NAME = 'Full Backup of AdventureWorks2012';
GO
```



Related Tasks

- [Back Up a Database \(SQL Server\)](#)
- [Create a Full Differential Backup \(SQL Server\)](#)
- [Restore a Database Backup \(SQL Server Management Studio\)](#)
- [Restore a Database Backup Under the Simple Recovery Model \(Transact-SQL\)](#)
- [Restore a Database to the Point of Failure Under the Full Recovery Model \(Transact-SQL\)](#)
- [Restore a Database to a New Location \(SQL Server\)](#)
- [Start the Maintenance Plan Wizard](#)



See Also

[Backup Overview \(SQL Server\)](#)

[Transaction Log Backups](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

[sp_addumpdevice](#)

[BACKUP](#)

[Back Up Database \(General Page\)](#)

[Back Up Database \(Options Page\)](#)

[Differential Backups \(SQL Server\)](#)

[Full Database Backups](#)

Back Up Database (General Page)

Use the **General** page of the **Back Up Database** dialog box to view or modify settings for a database back up operation.

For more information about basic backup concepts, see [Overview of the Recovery Models](#).

Note

When you specify a backup task by using SQL Server Management Studio, you can generate the corresponding Transact-SQL [BACKUP](#) script by clicking the **Script** button and then selecting a destination for the script.

To use SQL Server Management Studio to create a backup

- [How to: Back Up a Database \(SQL Server Management Studio\)](#)
- [How to: Create a Differential Database Backup \(SQL Server Management Studio\)](#)

Important

You can define a database maintenance plan to create database backups. For more information, see [Database Maintenance Plans](#) in SQL Server 2008 R2 Books Online.

To create a partial backup

- For a partial backup, you must use the Transact-SQL [BACKUP](#) statement with the PARTIAL option.

Options

Source

The options of the **Source** panel identify the database and specify the backup type and component for the back up operation.

Database

Select the database to back up.

Recovery model

View the recovery model (SIMPLE, FULL, or BULK_LOGGED) displayed for the selected database.

Backup type

Select the type of backup you want to perform on the specified database.

Backup type	Available for	Restrictions
Full	Databases, files, and filegroups	On the master database, only full backups are possible. Under the Simple Recovery

		Model, file and filegroup backups are available only for read-only filegroups.
Differential	Databases, files, and filegroups	Under the Simple Recovery Model, file and filegroup backups are available only for read-only filegroups.
Transaction Log	Transaction logs	Transaction log backups are not available for the Simple Recovery Model.

Copy Only Backup

Select to create a copy-only backup. A *copy-only backup* is a SQL Server backup that is independent of the sequence of conventional SQL Server backups. For more information, see [Copy-Only Backups](#).



Note

When the **Differential** option is selected, you cannot create a copy-only backup.

Backup component

Select the database component to be backed up. If **Transaction Log** is selected in the **Backup type** list, this option is not activated.

Select one of the following option buttons:

Database	Specifies that the entire database be backed up.
Files and filegroups	Specifies that the specified files and/or filegroups be backed up. Selecting this option, opens the Select Files and Filegroups dialog box. After you select the filegroups or files you want to back up and click Ok , your selections appear in the Filegroups and files box.

Backup set

The options of the **Backup set** panel allow for you to specify optional information about the backup set created by the back up operation.

Name

Specify the backup set name. The system automatically suggests a default name based on the database name and the backup type.

For information about backup sets, see [Media Sets, Media Families, and Backup](#)

[Sets.](#)

Description

Enter a description of the backup set.

Backup set will expire

Choose one of the following expiration options.

After	Specify the number of days that must elapse before this backup set expires and can be overwritten. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire. The default value for backup expiration is the value set in the Default backup media retention (in days) option. To access this, right-click the server name in Object Explorer and select Properties ; then click the Database Settings page of the Server Properties dialog box.
On	Specify a specific date when the backup set expires and can be overwritten.

Destination

The options of the **Destination** panel allow for you to specify the type of backup device for the back up operation and find an existing logical or physical backup device.

Note

For information about SQL Server backup devices, see [Backup Devices](#).

Back up to

Select one of the following types of media to which to back up. The destinations you select appear in the **Back up to** list.

Disk	Backs up to disk. This may be a system file or a disk-based logical backup device created for the database. The currently selected disks appear in the Back up to list. You can select up to 64 disk devices for your backup operation.
Tape	Backs up to tape. This may be a local tape

	<p>drive or a tape-based logical backup device created for the database. The currently selected tapes appear in the Back up to list. The maximum number is 64. If there are no tape devices attached to the server, this option is deactivated. The tapes you select are listed in the Back up to list.</p> <p>nNote</p> <p>Support for tape backup devices will be removed in a future version of SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.</p>
--	--

Add

Adds a file or device to the **Back up to** list. You can back up to 64 devices simultaneously on a local disk or remote disk. To specify a file on a remote disk, use the fully qualified universal naming convention (UNC) name. For more information, see [Backup Devices](#).

Remove

Removes one or more currently selected devices from the **Back up to** list.

Contents

Displays the media contents for the selected device.

See Also

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Define a Logical Backup Device for a Disk File \(SQL Server Management Studio\)](#)

[How to: Define a Logical Backup Device for a Tape Drive \(SQL Server Management Studio\)](#)

[Overview of the Recovery Models](#)

Back Up Database (Options Page)

Use the **Options** page of the **Back Up Database** dialog box to view or modify database backup options.

To create a backup by using SQL Server Management Studio

- [How to: Back Up a Database \(SQL Server Management Studio\)](#)

- [How to: Create a Differential Database Backup \(SQL Server Management Studio\)](#)

Important

You can define a database maintenance plan to create database backups. For more information, see [Maintenance Plans](#) and [How to: Start the Database Maintenance Plan Wizard](#).

Note

When you specify a backup task by using SQL Server Management Studio, you can generate the corresponding Transact-SQL [BACKUP](#) script by clicking the **Script** button and then selecting a destination for the script.

Options

Overwrite media

The options of the **Overwrite media** panel control how the backup is written to the media.

Note

For information about media sets, see [Media Sets, Media Families, and Backup Sets](#).

Back up to the existing media set

Back up a database to an existing media set. Selecting this option button activates three options.

Choose one of the following options:

Append to the existing backup set

Append the backup set to the existing media set, preserving any prior backups.

Overwrite all existing backup sets

Replace any prior backups on the existing media set with the current backup.

Check media set name and backup set expiration

Optionally, if backing up to an existing media set, require the backup operation to verify the name and the expiration date of the backup sets.

Media set name

If **Check media set name and backup set expiration** is selected, optionally, specify the name of the media set to be used for this backup operation.

Back up to a new media set, and erase all existing backup sets

Use a new media set, erasing the previous backup sets.

Clicking this option activates the following options:

New media set name

Optionally, enter a new name for the media set.

New media set description

Optionally, enter a meaningful description of the new media set. This description should be specific enough to communicate the contents accurately.

Reliability

The options of the **Transaction log** panel control error management by the backup operation.

Verify backup when finished

Verify that the backup set is complete and that all volumes are readable.

Perform checksum before writing to media

Verify the checksums before writing to the backup media. Selecting this option is equivalent to specifying the CHECKSUM option in the BACKUP statement of Transact-SQL. Selecting this option may increase the workload, and decrease the backup throughput of the backup operation. For information about backup checksums, see [Detecting and Coping with Media Errors](#).

Continue on error

The backup operation is to continue even after encountering one or more errors.

Transaction log

The options of the **Transaction log** panel control the behavior of a transaction log backup. These options are relevant only under the full recovery model or bulk-logged recovery model. They are activated only if **Transaction log** has been selected in the **Backup type** field on the General page of the **Back Up Database** dialog box.



Note

For information about transaction log backups, see [Transaction Log Backups \(SQL Server\)](#).

Truncate the transaction log

Back up the transaction log and truncate it to free log space. The database remains online. This is the default option.

Back up the tail of the log, and leave the database in the restoring state

Back up the tail of the log and leave the database in a restoring state. This option creates a *tail-log backup*, which backs up logs that have not yet been backed up (the active log), typically, in preparation for restoring a database. The database will be unavailable to users until it is completely restored.

Selecting this option is equivalent to specifying WITH NO_TRUNCATE, NORECOVERY in a [BACKUP](#) statement (Transact-SQL). For more information, see [Tail-Log Backups](#).

Tape drive

The options of the **Tape drive** panel control tape management during the backup operation. These options are activated only if **Tape** has been selected in the **Destination** panel on the General page of the **Back Up Database** dialog box.

 **Note**

For information about how to use tape devices, see [Backup Devices](#).

Unload the tape after backup

After the backup is complete, unload the tape.

Rewind the tape before unloading

Before unloading the tape, release and rewind it. This is enabled only if **Unload the tape after backup** is selected.

Compression

SQL Server 2008 Enterprise (or a later version) supports backup compression.

Set backup compression

In SQL Server 2008 Enterprise (or a later version), select one of the following backup compression values:

<p>Use the default server setting</p>	<p>Click to use the server-level default. This default is set by the backup compression default server-configuration option. For information about how to view the current setting of this option, see How to: View and Configure the backup compression default Option (SQL Server Management Studio).</p>
<p>Compress backup</p>	<p>Click to compress the backup, regardless of the server-level default.</p> <p>Important</p> <p>By default, compression significantly increases CPU usage, and the additional CPU consumed by the compression process might adversely impact concurrent operations. Therefore, you might want to create low-priority compressed backups in a session whose CPU usage is limited by Resource Governor. For more</p>

	information, see Use Resource Governor to Limit CPU Usage by Backup Compression (Transact-SQL) .
Do not compress backup	Click to create an uncompressed backup, regardless of the server-level default.

See Also

[BACKUP \(Transact-SQL\)](#)

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Back Up the Transaction Log When the Database Is Damaged \(Transact-SQL\)](#)

Select Backup Destination

Use the **Select Backup Destination** dialog box to select a device as your backup destination. A backup destination can be either a disk or a logical backup device.

To use SQL Server Management Studio to back up a database

- [Media Sets, Media Families, and Backup Sets](#)
- [How to: Create a Differential Database Backup \(SQL Server Management Studio\)](#)
- [How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)
- [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

Options

The options of this dialog box depend on whether you are selecting a destination on disk or on tape.

Destination on disk

To specify a backup destination, choose one of the following options.

File name	<p>Choose this option to enter a local or remote file as the backup destination in the text box.</p> <ul style="list-style-type: none"> • To specify a local file, click the browse button to the right of the text box and navigate to a file on the fixed drives of the computer running the server, or enter the full path and file name directly; for example, C:\Program Files\Microsoft SQL Server\MSSQL\Backup\AdventureWorksBackup.bak. • To specify a remote file as your backup destination, enter its fully qualified universal
------------------	--

	<p>naming convention (UNC) name. For more information, see Backup Devices.</p> <p> Important Backing up data over a network can be subject to network errors; therefore, we recommend that you verify the backup operation after it finishes. For more information, see RESTORE VERIFYONLY (Transact-SQL).</p>
Backup device	<p>Choose this option to select a logical backup device.</p> <p> Note For information about how to create a disk backup device, see How to: Create a Disk Backup Device (SQL Server Management Studio).</p>

Destination on tape

Specify a backup destination on a tape drive physically connected to the computer running the server (that is, the instance of the Database Engine). Choose one of the following options.

Tape drive	<p>Choose this option to select a tape drive as the backup destination from the list of tape drives that are physically connected to the computer that is running the server instance.</p> <p>Note Tape backup devices on remote computers are not valid backup destinations.</p>
Backup device	<p>Choose this option to select an existing logical backup device. These logical backup devices correspond to tape drives that are physically connected to the computer that is running the server instance.</p> <p> Note For information about how to</p>

	create a tape backup device, see How to: Create a Disk Backup Device (SQL Server Management Studio) .
--	---

See Also

[Backup Devices](#)

[Media Sets, Media Families, and Backup Sets](#)

Partial Backups

All SQL Server recovery models support partial backups, so this topic is relevant for all SQL Server databases. However, partial backups are designed for use under the simple recovery model to improve flexibility for backing up very large databases that contain one or more read-only filegroups.

Partial backups are useful whenever you want to exclude read-only filegroups. A *partial backup* resembles a full database backup, but a partial backup does not contain all the filegroups. Instead, for a read-write database, a partial backup contains the data in the primary filegroup, every read-write filegroup, and, optionally, one or more read-only files. A partial backup of a read-only database contains only the primary filegroup.

Note

If a read-only database is changed to read/write after a partial backup, there might be read/write secondary filegroups that are not in the partial backup. In this case, if you try to take a differential partial backup, the backup fails. Before you can take a differential partial backup of the database, you must take another partial backup. The new partial backup contains every read/write secondary filegroup and can serve as the base for differential partial backups.

File backups of read-only filegroups can be combined with partial backups. For information about file backups, see [Full File Backups](#).

A partial backup can serve as the *differential base* for differential partial backups. For more information, see [Using Differential Backups](#).

Related Tasks

Note

Partial backups are not supported by SQL Server Management Studio or the Maintenance Plan Wizard.

To create a partial backup

- [BACKUP \(Transact-SQL\)](#) (READ_WRITE_FILEGROUPS; FILEGROUP option, if needed)

To use a partial backup in a restore sequence

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)

See Also

[Backup Overview \(SQL Server\)](#)

[File Restores \(Simple Recovery Model\)](#)

[Piecemeal Restores](#)

Full File Backups

This topic is relevant for SQL Server databases that contain multiple files or filegroups.

The files in a SQL Server database can be backed up and restored individually. Also, you can specify a whole filegroup instead of specifying each constituent file individually. Note that if any file in a filegroup is offline (for example, because the file is being restored), the whole filegroup is offline and cannot be backed up.

File backups of read-only filegroups can be combined with partial backups. Partial backups include all the read/write filegroups and, optionally, one or more read-only filegroups. For more information, see [Partial Backups](#).

A file backup can serve as the *differential base* for differential file backups. For more information, see [Differential Backups](#).



Note

Full file backups are typically called *file backups*, except when they are being explicitly compared with *differential file backups*.

In This Topic:

- Benefits of File Backups
- Disadvantages of File Backups
- Overview of File Backups
- Related Tasks

Benefits of File Backups

File backups offer the following advantages over database backups:

- Using file backups can increase the speed of recovery by letting you restore only damaged files, without restoring the rest of the database.

For example, if a database consists of several files that are located on different disks and one disk fails, only the file on the failed disk has to be restored. The damaged file can be quickly restored, and recovery is faster than it would be for an entire database.

- File backups increase flexibility in scheduling and media handling over full database backups, which for very large databases can become unmanageable. The increased flexibility of file or filegroup backups is also useful for large databases that contain data that has varying update characteristics.

Disadvantages of File Backups

- The primary disadvantage of file backups compared to full database backups is the additional administrative complexity. Maintaining and keeping track of a complete set of these backups can be a time-consuming task that might outweigh the space requirements of full database backups.
- A media failure can make a complete database unrecoverable if a damaged file lacks a backup. You must therefore maintain a complete set of file backups, and, for the full/bulk-logged recovery model, one or more log backups covering minimally the interval between the first full file backup and last full file backup.

Overview of File Backups

A full file backup backs up all the data in one or more files or filegroups. By default, file backups contain enough log records to roll forward the file to the end of the backup operation.

Backing up a read-only file or filegroup is the same for every recovery model. Under the full recovery model, a complete set of full file backups, together with enough log backups to span all the file backups, is the equivalent of a full database backup.

Only one file backup operation can occur at a time. You can back up multiple files in one operation, but this might extend the recovery time if you only have to restore a single file. This is because to locate that file, the whole backup is read.



Note

Individual files can be restored from a database backup; however, locating and restoring a file takes longer from a database backup than from a file backup.

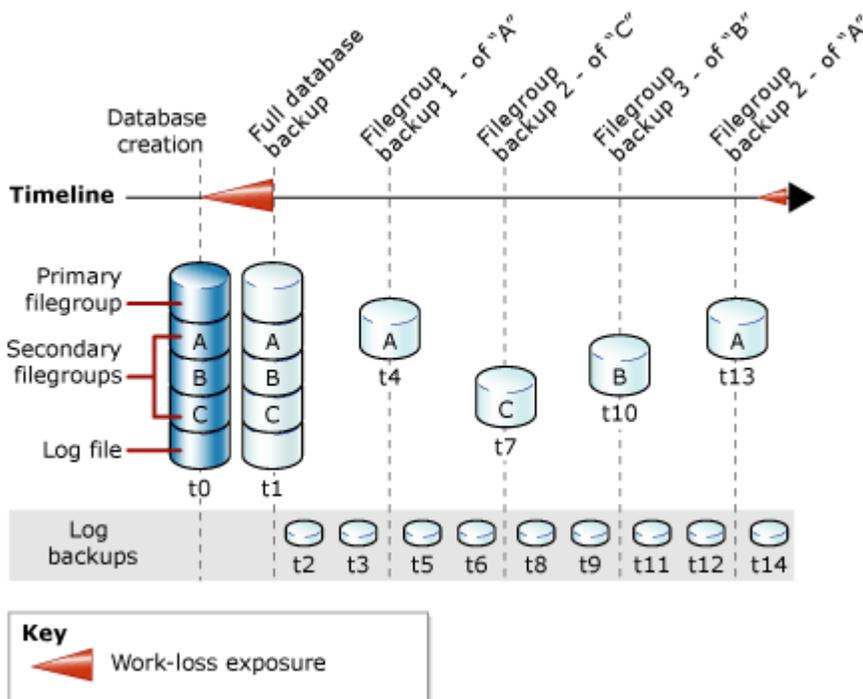
File Backups and the Simple Recovery Model

Under the simple recovery model, read/write files must all be backed up together. This makes sure that the database can be restored to a consistent point in time. Instead of individually specifying each read/write file or filegroup, use the `READ_WRITE_FILEGROUPS` option. This option backs up all the read/write filegroups in the database. A backup that is created by specifying `READ_WRITE_FILEGROUPS` is known as a partial backup. For more information, see [Partial Backups](#).

File Backups and the Full Recovery Model

Under the full recovery model, you must back up the transaction log, regardless of the rest of your backup strategy. A complete set of full file backups, together with enough log backups to span all the file backups from the start of the first file backup, is the equivalent of a full database backup.

Restoring a database using just file and log backups can be complex. Therefore, if it is possible, it is a best practice to perform a full database backup and start the log backups before the first file backup. The following illustration shows a strategy in which a full database backup is taken (at time t1) soon after the database is created (at time t0). This first database backup enables transaction log backups to start. Transaction log backups are scheduled to occur at set intervals. File backups occur at whatever interval best meets the business requirements for the database. This illustration shows each of the four filegroups being backed up one at a time. The order in which they are backed up (A, C, B, A) reflects the business requirements of the database.



Note

Under the full recovery model, you must roll forward the transaction log when restoring a read/write file backup to make sure that the file is consistent with the rest of the database. To avoid rolling forward a lot of transaction log backups, consider using differential file backups. For more information, see [Differential Backups](#).

Related Tasks

To create a file or filegroup backup

- [Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Backup.SqlBackup(Microsoft.SqlServer.Management.Smo.Server) (SMO)



Note

File backups are not supported by the Maintenance Plan Wizard.

See Also

[BACKUP \(Transact-SQL\)](#)

[Backup Overview \(SQL Server\)](#)

[Backup and Restore Considerations for Related Features](#)

[Differential Backups](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[Performing Online Restores](#)

[Performing Piecemeal Restores](#)

Back Up Files and Filegroups

This topic describes how to back up files and filegroups in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. When the database size and performance requirements make a full database backup impractical, you can create a file backup instead. A *file backup* contains all the data in one or more files (or filegroups). For more information about file backups, see [RESTORE \(Transact-SQL\)](#) and [Differential Backups \(SQL Server\)](#).

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Recommendations
 - Security
- **To back up files and filegroups, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The BACKUP statement is not allowed in an explicit or implicit transaction.
- Under the simple recovery model, read/write files must all be backed up together. This helps make sure that the database can be restored to a consistent point in time. Instead of individually specifying each read/write file or filegroup, use the READ_WRITE_FILEGROUPS option. This option backs up all the read/write filegroups in the database. A backup that is created by specifying READ_WRITE_FILEGROUPS is known as a *partial backup*. For more information, see [Partial Backups](#).
- For more information about limitations and restrictions, see [Backup Overview \(SQL Server\)](#).

Recommendations

- By default, every successful backup operation adds an entry in the SQL Server error log and in the system event log. If you back up the log very frequently, these success messages accumulate quickly, resulting in huge error logs that can make finding other messages difficult. In such cases you can suppress these log entries by using trace flag 3226 if none of your scripts depend on those entries. For more information, see [Trace Flags \(Transact-SQL\)](#).

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

▶ To back up database files and filegroups

1. After connecting to the appropriate instance of the SQL Server Database Engine,

in Object Explorer, click the server name to expand the server tree.

2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Database** list, verify the database name. You can optionally select a different database from the list.
5. In the **Backup type** list, select **Full** or **Differential**.
6. For the **Backup component** option, click **File and Filegroups**.
7. In the **Select Files and Filegroups** dialog box, select the files and filegroups you want to back up. You can select one or more individual files or check the box for a filegroup to automatically select all the files in that filegroup.
8. Either accept the default backup set name suggested in the **Name** text box, or enter a different name for the backup set.
9. Optionally, in the **Description** text box, enter a description of the backup set.
10. Specify when the backup set will expire:
 - To have the backup set expire after a specific number of days, click **After** (the default option). Then, enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.
The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (**Database Settings** page). To access this option, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.
 - To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.
11. Choose the type of backup destination by clicking **Disk** or **Tape**. To select the paths of up to 64 disk or tape drives that contain a single media set, click **Add**. The selected paths are displayed in the **Backup to** list.



Note

To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.

12. To view or select the advanced options, click **Options** in the **Select a page** pane.
13. Select an **Overwrite Media** option, by clicking one of the following:
 - **Back up to the existing media set**
For this option, click either **Append to the existing backup set** or **Overwrite all existing backup sets**. For information about backing up to an existing media set, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

Optionally, select **Check media set name and backup set expiration** to cause the backup operation to verify the date and time at which the media set and backup set expire.

Optionally, enter a name in the **Media set name** text box. If no name is specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see whether the actual name matches the name that you enter here.

If you leave the media name blank and check the box to check it against the media, success will equal the media name on the media also being blank.

- **Back up to a new media set, and erase all existing backup sets**

For this option, enter a name in the **New media set name** text box, and, optionally, describe the media set in the **New media set description** text box. For more information about creating a new media set, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

14. In the **Reliability** section, optionally check:

- **Verify backup when finished.**
- **Perform checksum before writing to media**, and, optionally, **Continue on checksum error**. For more information about checksums, see [Detecting and Coping with Media Errors](#).

15. If you are backing up to a tape drive (as specified in the **Destination** section of the **General** page), the **Unload the tape after backup** option is active. Clicking this option enables the **Rewind the tape before unloading** option.



Note

The options in the **Transaction log** section are inactive unless you are backing up a transaction log (as specified in the **Backup type** section of the **General** page).

16. SQL Server 2008 Enterprise and later versions support [backup compression](#). By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by checking **Compress backup**, and you can prevent compression by checking **Do not compress backup**.

To view the current backup compression default

- [How to: View and Configure the backup compression default Option \(SQL Server Management Studio\)](#)



Using Transact-SQL

► To back up files and filegroups

- To create a file or filegroup backup, use a [BACKUP DATABASE <file or filegroup>](#) statement. Minimally, this statement must specify the following:
 - The database name.
 - A FILE or FILEGROUP clause for each file or filegroup, respectively.
 - The backup device on which the full backup will be written.

The basic Transact-SQL syntax for a file backup is:

```
BACKUP DATABASE database
  { FILE = logical_file_name | FILEGROUP = logical_filegroup_name } [ ,...f ]
  TO backup_device [ ,...n ]
  [ WITH with_options [ ,...o ] ] ;
```

Option	Description
database	Is the database from which the transaction log, partial database, or complete database is backed up.
FILE = <i>logical_file_name</i>	Specifies the logical name of a file to include in the file backup.
FILEGROUP = <i>logical_filegroup_name</i>	Specifies the logical name of a filegroup to include in the file backup. Under the simple recovery model, a filegroup backup is allowed only for a read-only filegroup.
[,...f]	Is a placeholder that indicates that multiple files and filegroups may be specified. The number of files or filegroups is unlimited.
backup_device [,...n]	Specifies a list of from 1 to 64 backup devices to use for the backup operation. You can specify a physical backup device, or you can specify a corresponding logical backup device, if already defined. To specify a physical backup device, use the DISK or TAPE option: { DISK TAPE } =

	<p>physical_backup_device_name</p> <p>For more information, see Backup Devices.</p>
<p>WITH with_options [,...o]</p>	<p>Optionally, specifies one or more additional options, such as DIFFERENTIAL.</p> <p>nNote</p> <p>A differential file backup requires a full file backup as a base. For more information, see Create a Differential Database Backup (SQL Server).</p>

- Under the full recovery model, you must also back up the transaction log. To use a complete set of full file backups to restore a database, you must also have enough log backups to span all the file backups, from the start of the first file backup. For more information, see [Back Up a Transaction Log](#).

Examples (Transact-SQL)

The following examples back up one or more files of the secondary filegroups of the `Sales` database. This database uses the full recovery model and contains the following secondary filegroups:

- A filegroup named `SalesGroup1` that has the files `SGrp1Fi1` and `SGrp1Fi2`.
- A filegroup named `SalesGroup2` that has the files `SGrp2Fi1` and `SGrp2Fi2`.

A. Creating a file backup of two files

The following example creates a differential file backup of only the `SGrp1Fi2` file of the `SalesGroup1` and the `SGrp2Fi2` file of the `SalesGroup2` filegroup.

```
--Backup the files in the SalesGroup1 secondary filegroup.
BACKUP DATABASE Sales
    FILE = 'SGrp1Fi2',
    FILE = 'SGrp2Fi2'
    TO DISK = 'G:\SQL Server Backups\Sales\SalesGroup1.bck';
GO
```

B. Creating a full file backup of the secondary filegroups

The following example creates a full file backup of every file in both of the secondary filegroups.

```
--Back up the files in SalesGroup1.  
BACKUP DATABASE Sales  
    FILEGROUP = 'SalesGroup1',  
    FILEGROUP = 'SalesGroup2'  
    TO DISK = 'C:\MySQLServer\Backups\Sales\SalesFiles.bck';  
GO
```

C. Creating a differential file backup of the secondary filegroups

The following example creates a differential file backup of every file in both of the secondary filegroups.

```
--Back up the files in SalesGroup1.  
BACKUP DATABASE Sales  
    FILEGROUP = 'SalesGroup1',  
    FILEGROUP = 'SalesGroup2'  
    TO DISK = 'C:\MySQLServer\Backups\Sales\SalesFiles.bck'  
    WITH  
        DIFFERENTIAL;  
GO
```



See Also

[Backup Overview \(SQL Server\)](#)

[BACKUP](#)

[RESTORE \(Transact-SQL\)](#)

[Viewing Information About Backups](#)

[Back Up Database \(General Page\)](#)

[Back Up Database \(Options Page\)](#)

[Full File Backups](#)

[Differential Backups \(SQL Server\)](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

Differential Backups

This backup and restore topic is relevant for all SQL Server databases.

A differential backup is based on the most recent, previous full data backup. A differential backup captures only the data that has changed since that full backup. The full backup upon which a differential backup is based is known as the *base* of the differential. Full backups, except for copy-only backups, can serve as the base for a series of differential backups, including database backups, partial backups, and file backups. The base backup for a file differential backup can be contained within a full backup, a file backup, or a partial backup.

In this Topic:

- Benefits
- Overview of Differential Backups
- Differential Backups of Read-Only Databases
- Related Tasks

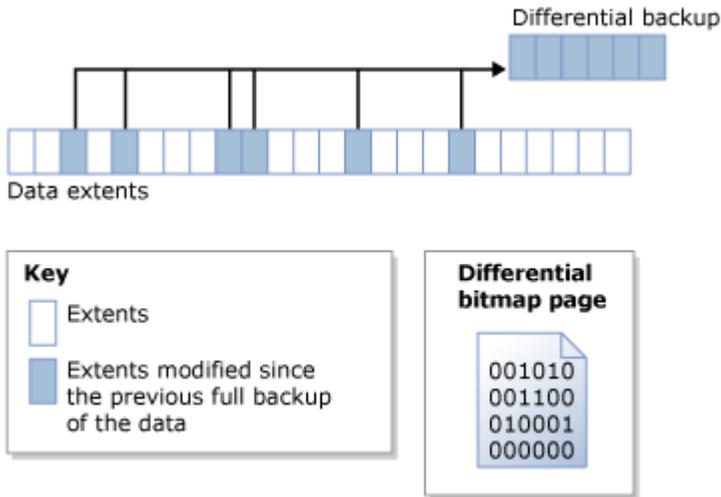
Benefits

- Creating a differential backups can be very fast compared to creating a full backup. A differential backup records only the data that has changed since the full backup upon the differential backup is based. This facilitates taking frequent data backups, which decrease the risk of data loss. However, before you restore a differential backup, you must restore its base. Therefore restoring from a differential backup will necessarily take more steps and time than restoring from a full backup because two backup files are required.
- Differential database backups are especially useful if a subset of a database is modified more frequently than the rest of the database. In these cases, differential database backups enable you back up frequently without the overhead of full database backups.
- Under the full recovery model, using differential backups can reduce the number of log backups that you have to restore.

Overview of Differential Backups

A differential backup captures the state of any *extents* (collections of eight physically contiguous pages) that have changed between when the differential base was created and the when differential backup is created. This means that the size of a given differential backup depends on the amount of data that has changed since the base. Generally, the older a base is, the larger a new differential backup will be. In a series of differential backups, a frequently updated extent is likely to contain different data in each differential backup.

The following illustration shows how a differential backup works. The figure shows 24 data extents, 6 of which have changed. The differential backup contains only these 6 data extents. The differential backup operation relies on a bitmap page that contains a bit for every extent. For each extent updated since the base, the bit is set to 1 in the bitmap.



 **Note**

The differential bitmap is not updated by a copy-only backup. Therefore, a copy-only backup does not affect subsequent differential backups.

A differential backup that is taken fairly soon after its base is usually significantly smaller than the differential base. This saves storage space and backup time. However, as a database changes over time, the difference between the database and a specific differential base increases. The longer the time between a differential backup and its base, the larger the differential backup is likely to be. This means that the differential backups can eventually approach the differential base in size. A large differential backup loses the advantages of a faster and smaller backup.

As the differential backups increase in size, restoring a differential backup can significantly increase the time that is required to restore a database. Therefore, we recommend that you take a new full backup at set intervals to establish a new differential base for the data. For example, you might take a weekly full backup of the whole database (that is, a full database backup) followed by a regular series of differential database backups during the week.

At restore time, before you restore a differential backup, you must restore its base. Then, restore only the most recent differential backup to bring the database forward to the time when that differential backup was created. Typically, you would restore the most recent full backup followed by the most recent differential backup that is based on that full backup.

Differential Backups of Read-Only Databases

For read-only databases, full backups used alone are easier to manage than when they are used with differential backups. When a database is read-only, backup and other operations cannot change the metadata that is contained in the file. Therefore, metadata that is required by a differential backup, such as the log sequence number at which the

differential backup begins (the differential base LSN) is stored in the **master** database. If the differential base is taken when the database is read-only, the differential bitmap indicates more changes than have actually occurred since the base backup. The extra data is read by backup, but is not written to the backup, because the **differential_base_lsn** stored in the [backupset](#) system table is used to determine whether the data has actually changed since the base.

When a read-only database is rebuilt, restored, or detached and attached, the differential-base information is lost. This occurs because the **master** database is not synchronized with the user database. The SQL Server Database Engine cannot detect or prevent this problem. Any later differential backups are not based on the most recent full backup and could provide unexpected results. To establish a new differential base, we recommend that you create a full database backup.

Best Practices for Using Differential Backups with a Read-Only Database

After you create a full database backup of a read-only database, if you intend to create a subsequent differential backup, back up the **master** database.

If the **master** database is lost, restore it before you restore any differential backup of a user database.

If you detach and attach a read-only database for which you plan to later use differential backups, as soon as it is practical, take a full database backup of both the read-only database and of the **master** database.

Related Tasks

- [Create A Differential Database Backup \(SQL Server\)](#)
- [Restore a Differential Database Backup \(SQL Server\)](#)

See Also

[Backup Overview](#)

[Full Database Backups](#)

[Complete Database Restores \(Full Recovery Model\)](#)

[Complete Database Restores \(Simple Recovery Model\)](#)

[Create Transaction Log Backups \(SQL Server\)](#)

Create a Differential Database Backup

This topic describes how to create a differential database backup in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**

Limitations and Restrictions

Prerequisites

Recommendations

Security

- **To create a differential database backup, using:**

SQL Server Management Studio

Transact-SQL

Before You Begin

Limitations and Restrictions

- The BACKUP statement is not allowed in an explicit or implicit transaction.

Prerequisites

- Creating a differential database backup requires that a previous full database backup exist. If the selected database has never been backed up, run a full database backup before creating any differential backups. For more information, see [Differential Database Backups](#).

Recommendations

- As the differential backups increase in size, restoring a differential backup can significantly increase the time that is required to restore a database. Therefore, we recommend that you take a new full backup at set intervals to establish a new differential base for the data. For example, you might take a weekly full backup of the whole database (that is, a full database backup) followed by a regular series of differential database backups during the week.

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's

physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

▶ To create a differential database backup

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Database** list box, verify the database name. You can optionally select a different database from the list.

You can perform a differential backup for any recovery model (full, bulk-logged, or simple).

5. In the **Backup type** list box, select **Differential**.

Important

When **Differential** is selected, verify that the **Copy Only Backup** check box is cleared.

6. For **Backup component**, click **Database**.
7. Either accept the default backup set name suggested in the **Name** text box, or enter a different name for the backup set.
8. Optionally, in the **Description** text box, enter a description of the backup set.
9. Specify when the backup set will expire:
 - To have the backup set expire after a specific number of days, click **After** (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.
The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (**Database Settings** page). To access this, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.
 - To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.

10. Choose the type of backup destination by clicking **Disk** or **Tape**. To select the path of up to 64 disk or tape drives containing a single media set, click **Add**. The

selected paths are displayed in the **Backup to** list box.

To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.

11. To view or select the advanced options, click **Options** in the **Select a page** pane.

12. Select an **Overwrite Media** option, by clicking one of the following:

- **Back up to the existing media set**

For this option, click either **Append to the existing backup set** or **Overwrite all existing backup sets**. Optionally, check the **Check media set name and backup set expiration** check box and, optionally, enter a name in the **Media set name** text box. If no name is specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see if the actual name matches the name you enter here.

If you leave the media name blank and check the box to check it against the media, success will equal the media name on the media also being blank.

- **Back up to a new media set, and erase all existing backup sets**

For this option, enter a name in the **New media set name** text box, and, optionally, describe the media set in the **New media set description** text box.

13. In the **Reliability** section, optionally, check:

- **Verify backup when finished.**

- **Perform checksum before writing to media**, and, optionally, **Continue on checksum error**. For information about checksums, see [Detecting and Coping with Media Errors](#).

14. If you are backing up to a tape drive (as specified in the **Destination** section of the **General** page), the **Unload the tape after backup** option is active. Clicking this option activates the **Rewind the tape before unloading** option.



Note

The options in the **Transaction log** section are inactive unless you are backing up a transaction log (as specified in the **Backup type** section of the **General** page).

15. SQL Server 2008 Enterprise and later supports [backup compression](#). By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by checking **Compress backup**, and you can prevent compression by checking **Do not compress backup**.

To view the current backup compression default

- [How to: View and Configure the backup compression default Option \(SQL Server Management Studio\)](#)



Note

Alternatively, you can use the Maintenance Plan Wizard to create differential database backups.



Using Transact-SQL

▶ To create a differential database backup

1. Execute the BACKUP DATABASE statement to create the differential database backup, specifying:
 - The name of the database to back up.
 - The backup device where the full database backup is written.
 - The DIFFERENTIAL clause, to specify that only the parts of the database that have changed after the last full database backup was created are backed up.

The required syntax is:

```
BACKUP DATABASE database_name TO <backup_device> WITH DIFFERENTIAL
```

Example (Transact-SQL)

This example creates a full and a differential database backup for the `MyAdvWorks` database.

```
-- Create a full database backup first.
```

```
BACKUP DATABASE MyAdvWorks
```

```
    TO MyAdvWorks_1
```

```
    WITH INIT;
```

```
GO
```

```
-- Time elapses.
```

```
-- Create a differential database backup, appending the backup
```

```
-- to the backup device containing the full database backup.
```

```
BACKUP DATABASE MyAdvWorks
```

```
    TO MyAdvWorks_1
```

```
    WITH DIFFERENTIAL;
```

```
GO
```



See Also

[Differential Backups \(SQL Server\)](#)

[How to: Back Up a Database \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Restore a Full Differential Backup \(SQL Server Management Studio\)](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[Maintenance Plans](#)

[Using File Backups](#)

Restore a Differential Database Backup

This topic describes how to restore a differential database backup in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Prerequisites
 - Security
- **To restore a differential database backup, using:**
 - SQL Server Management Studio
 - Transact-SQL
- Related Tasks

Before You Begin

Limitations and Restrictions

- RESTORE is not allowed in an explicit or implicit transaction.
- Backups that are created by more recent version of SQL Server cannot be restored in earlier versions of SQL Server.
- In SQL Server 2012, you can restore a user database from a database backup that was created by using SQL Server 2005 or a later version. However, backups of **master**, **model** and **msdb** that were created by using SQL Server 2005 or SQL Server 2008 cannot be restored by SQL Server 2012.

Prerequisites

- Under the full or bulk-logged recovery model, before you can restore a database, you must back up the active transaction log (known as the tail of the log). For more information, see [Back Up a Transaction Log](#).

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore a differential database backup

1. After you connect to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases**, and then select a system database.
3. Right-click the database, point to **Tasks**, point to **Restore**, and then click **Database**.
4. On the **General** page, use the **Source** section to specify the source and location of the backup sets to restore. Select one of the following options:

- **Database**

Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.



Note

If the backup is taken from a different server, the destination server will not have the backup history information for the specified database. In this case, select **Device** to manually specify the file or device to restore.

- **Device**

Click the browse (...) button to open the **Select backup devices** dialog box. In the **Backup media type** box, select one of the listed device types. To select one or more devices for the **Backup media** box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.

In the **Source: Device: Database** list box, select the name of the database which should be restored.

Note This list is only available when **Device** is selected. Only databases that have backups on the selected device will be available.

5. In the **Destination** section, the **Database** box is automatically populated with the name of the database to be restored. To change the name of the database, enter the new name in the **Database** box.



Note

To stop the restore at a specific point in time, click **Timeline** to access the **Backup Timeline** dialog box. For help with stopping a database restore at a specific point in time, see [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).

6. In the **Backup sets to restore** grid, select the backups through the differential backup that you wish to restore.

For information about the columns in the **Backup sets to restore** grid, see [SQL Server Management Studio Tutorial](#).

7. On the **Options** page, in the **Restore options** panel, you can select any of the following options, if appropriate for your situation:

- **Overwrite the existing database (WITH REPLACE)**
- **Preserve the replication settings (WITH KEEP_REPLICATION)**
- **Prompt before restoring each backup**
- **Restrict access to the restored database (WITH RESTRICTED_USER)**

For more information about these options, see [Restore Database \(Options Page\)](#).

8. Select an option for the **Recovery state** box. This box determines the state of the database after the restore operation.

- **RESTORE WITH RECOVERY** is the default behavior which leaves the database ready for use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored. Select this option if you are restoring all of the necessary backups now.
- **RESTORE WITH NORECOVERY** which leaves the database non-operational, and does not roll back the uncommitted transactions. Additional transaction logs can be restored. The database cannot be used until it is recovered.
- **RESTORE WITH STANDBY** which leaves the database in read-only mode. It undoes uncommitted transactions, but saves the undo actions in a standby file so that recovery effects can be reverted.

For descriptions of the options, see [Restore Database \(Options Page\)](#).

9. Restore operations will fail if there are active connections to the database. Check the **Close existing connections option** to ensure that all active connections

between Management Studio and the database are closed.

10. Select **Prompt before restoring each backup** if you wish to be prompted between each restore operation. This is not usually necessary unless the database is large and you wish to monitor the status of the restore operation.
11. Optionally, use the **Files** page to restore the database to a new location. For help with moving a database, see [Restore a Database to a New Location \(SQL Server Management Studio\)](#).
12. Click .



Using Transact-SQL

▶ To restore a differential database backup

1. Execute the RESTORE DATABASE statement, specifying the NORECOVERY clause, to restore the full database backup that comes before the differential database backup. For more information, see [How to: Restore a Full Backup](#).
2. Execute the RESTORE DATABASE statement to restore the differential database backup, specifying:
 - The name of the database to which the differential database backup is applied.
 - The backup device where the differential database backup is restored from.
 - The NORECOVERY clause if you have transaction log backups to apply after the differential database backup is restored. Otherwise, specify the RECOVERY clause.
3. With the full or bulk-logged recovery model, restoring a differential database backup restores the database to the point at which the differential database backup was completed. To recover to the point of failure, you must apply all transaction log backups created after the last differential database backup was created. For more information, see [Apply Transaction Log Backups \(SQL Server\)](#).

Examples (Transact-SQL)

A. Restoring a differential database backup

This example restores a database and differential database backup of the MyAdvWorks database.

```
-- Assume the database is lost, and restore full database,  
-- specifying the original full database backup and NORECOVERY,  
-- which allows subsequent restore operations to proceed.  
RESTORE DATABASE MyAdvWorks
```

```

    FROM MyAdvWorks_1
    WITH NORECOVERY;
GO
-- Now restore the differential database backup, the second backup on
-- the MyAdvWorks_1 backup device.
RESTORE DATABASE MyAdvWorks
    FROM MyAdvWorks_1
    WITH FILE = 2,
    RECOVERY;

```

```
GO
```

B. Restoring a database, differential database, and transaction log backup

This example restores a database, differential database, and transaction log backup of the MyAdvWorks database.

```

-- Assume the database is lost at this point. Now restore the full
-- database. Specify the original full database backup and NORECOVERY.
-- NORECOVERY allows subsequent restore operations to proceed.

```

```

RESTORE DATABASE MyAdvWorks
    FROM MyAdvWorks_1
    WITH NORECOVERY;

```

```
GO
```

```

-- Now restore the differential database backup, the second backup on
-- the MyAdvWorks_1 backup device.

```

```

RESTORE DATABASE MyAdvWorks
    FROM MyAdvWorks_1
    WITH FILE = 2,
    NORECOVERY;

```

```
GO
```

```

-- Now restore each transaction log backup created after
-- the differential database backup.

```

```

RESTORE LOG MyAdvWorks
    FROM MyAdvWorks_log1
    WITH NORECOVERY;

```

```
GO
```

```
RESTORE LOG MyAdvWorks
    FROM MyAdvWorks_log2
    WITH RECOVERY;

GO
```

Related Tasks

- [Create a Differential Database Backup \(SQL Server\)](#)
- [Restore a Transaction Log Backup \(SQL Server\)](#)



See Also

[Differential Backups \(SQL Server\)](#)

[RESTORE](#)

Copy-Only Backups

A *copy-only backup* is a SQL Server backup that is independent of the sequence of conventional SQL Server backups. Usually, taking a backup changes the database and affects how later backups are restored. However, occasionally, it is useful to take a backup for a special purpose without affecting the overall backup and restore procedures for the database. Copy-only backups serve this purpose. The transaction log is never truncated after a copy-only backup.

The types of copy-only backups are as follows:

- Copy-only full backups (all recovery models)
A copy-only full backup cannot serve as a differential base or differential backup and does not affect the differential base.
- Copy-only log backups (full recovery model and bulk-logged recovery model only)
A copy-only log backup preserves the existing log archive point and, therefore, does not affect the sequencing of regular log backups. A copy-only log backup can sometimes be useful for performing an online restore. For an example of this, see [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#).



Important

Typically, copy-only log backups are unnecessary. Instead, you can create a regular log backup (using WITH NORECOVERY), and use that backup together with all other previous log backups that are required for the restore sequence.

Restoring a copy-only full backup is the same as restoring any full backup.

Related Tasks

To create a copy-only backup (Transact-SQL)

- [How to: Back Up a Database \(SQL Server Management Studio\)](#) (Copy Only Backup option)
- [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#) (Copy Only Backup option)
- [BACKUP \(Transact-SQL\)](#) (COPY_ONLY option)

To view copy-only backups

- [backupset \(Transact-SQL\)](#) (is_copy_only column)

See Also

[BACKUP \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[Backup Overview \(SQL Server\)](#)

[Recovery Models](#)

[Copying Databases with Backup and Restore](#)

[Restore and Recovery Overview \(SQL Server\)](#)

Transaction Log Backups

This topic is relevant only for SQL Server databases that are using the full or bulk-logged recovery models. This topic discusses backing up the transaction log of a SQL Server database.

Minimally, you must have created at least one full backup before you can create any log backups. After that, the transaction log can be backed up at any time unless the log is already being backed up. We recommend that you take log backups frequently, both to minimize work loss exposure and to truncate the transaction log. Typically, a database administrator creates a full database backup occasionally, such as weekly, and, optionally, creates a series of differential database backup at a shorter interval, such as daily. Independently of the database backups, the database administrator backs up the transaction log at frequent intervals, such as every 10 minutes. For a given type of backup, the optimal interval depends on factors such as the importance of the data, the size of the database, and the workload of the server.

In this Topic:

- How a Sequence of Log Backups Works
- Recommendations

- Related Tasks
- Related Content

How a Sequence of Log Backups Works

The sequence of transaction log backups *log chain* is independent of data backups. For example, assume the following sequence of events.

Time	Event
8:00 A.M.	Back up database.
Noon	Back up transaction log.
4:00 P.M.	Back up transaction log.
6:00 P.M.	Back up database.
8:00 P.M.	Back up transaction log.

The transaction log backup created at 8:00 P.M. contains transaction log records from 4:00 P.M. through 8:00 P.M., spanning the time when the full database backup was created at 6:00 P.M. The sequence of transaction log backups is continuous from the initial full database backup created at 8:00 A.M. to the last transaction log backup created at 8:00 P.M. For information about how to apply these log backups, see the example in [Applying Transaction Log Backups](#).

Recommendations

- If a transaction log is damaged, work that is performed since the most recent valid backup is lost. Therefore we strongly recommend that you put your log files on fault-tolerant storage.
- If a database is damaged or you are about to restore the database, we recommend that you create a tail-log backup to enable you to restore the database to the current point in time.
- By default, every successful backup operation adds an entry in the SQL Server error log and in the system event log. If back up the log very frequently, these success messages accumulate quickly, resulting in huge error logs that can make finding other messages difficult. In such cases you can suppress these log entries by using trace flag 3226 if none of your scripts depend on those entries. For more information, see [Trace Flags \(Transact-SQL\)](#).

Related Tasks

To create a transaction log backup

- [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Backup.SqlBackup(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To schedule backup jobs, see [Use the Maintenance Plan Wizard](#).

Related Content

None.

See Also

[Transaction Logs \(SQL Server\)](#)

[Back Up and Restore of SQL Server Databases](#)

[Tail-Log Backups \(SQL Server\)](#)

[Apply Transaction Log Backups \(SQL Server\)](#)

Back Up a Transaction Log

This topic describes how to back up a transaction log in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Recommendations
 - Security
- **To back up a transaction log, using:**
 - SQL Server Management Studio
 - Transact-SQL



Note

Alternatively, you can use the [Maintenance Plan Wizard](#) to create backups.

Before You Begin

Limitations and Restrictions

- The BACKUP statement is not allowed in an explicit or implicit transaction.

Recommendations

- If a database uses either the full or bulk-logged recovery model, you must back up the transaction log regularly enough to protect your data and to keep the transaction log from filling. This truncates the log and supports restoring the database to a specific point in time.
- By default, every successful backup operation adds an entry in the SQL Server error log and in the system event log. If back up the log very frequently, these success messages accumulate quickly, resulting in huge error logs that can make finding other messages difficult. In such cases you can suppress these log entries by using trace flag 3226 if none of your scripts depend on those entries. For more information, see [Trace Flags \(Transact-SQL\)](#).

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

To back up a transaction log

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Database** list box, verify the database name. You can optionally select a

different database from the list.

5. Verify that the recovery model is either **FULL** or **BULK_LOGGED**.
6. In the **Backup type** list box, select **Transaction Log**.
7. Optionally, you can select **Copy Only Backup** to create a copy-only backup. A *copy-only backup* is a SQL Server backup that is independent of the sequence of conventional SQL Server backups. For more information, see [Copy-Only Backups](#).



Note

When the **Differential** option is selected, you cannot create a copy-only backup.

8. Either accept the default backup set name suggested in the **Name** text box, or enter a different name for the backup set.
9. Optionally, in the **Description** text box, enter a description of the backup set.
10. Specify when the backup set will expire:
 - To have the backup set expire after a specific number of days, click **After** (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.

The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (**Database Settings** page). To access this dialog box, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.
 - To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.
11. Choose the type of backup destination by clicking **Disk** or **Tape**. To select the paths of up to 64 disk or tape drives containing a single media set, click **Add**. The selected paths are displayed in the **Backup to** list box.

To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.
12. To view or select the advanced options, click **Options** in the **Select a page** pane.
13. Select an **Overwrite Media** option, by clicking one of the following:
 - **Back up to the existing media set**

For this option, click either **Append to the existing backup set** or **Overwrite all existing backup sets**. For more information, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

Optionally, select **Check media set name and backup set expiration** to cause the backup operation to verify the date and time at which the media set and backup set expire.

Optionally, enter a name in the **Media set name** text box. If no name is

specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see whether the actual name matches the name you enter here.

If you leave the media name blank and check the box to check it against the media, success will equal the media name on the media also being blank.

- **Back up to a new media set, and erase all existing backup sets**

For this option, enter a name in the **New media set name** text box, and, optionally, describe the media set in the **New media set description** text box. For more information, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

14. In the **Reliability** section, optionally, check:

- **Verify backup when finished.**
- **Perform checksum before writing to media**, and, optionally, **Continue on checksum error**. For information on checksums, see [Detecting and Coping with Media Errors](#).

15. In the **Transaction log** section:

- For routine log backups, keep the default selection, **Truncate the transaction log by removing inactive entries**.
- To back up the tail of the log (that is, the active log), check **Back up the tail of the log, and leave database in the restoring state**.

A tail-log backup is taken after a failure to back up the tail of the log in order to prevent work loss. Back up the active log (a tail-log backup) both after a failure, before beginning to restore the database, or when failing over to a secondary database. Selecting this option is equivalent to specifying the NORECOVERY option in the BACKUP LOG statement of Transact-SQL. For more information about tail-log backups, see [Overview of Tail-Log Backups](#).

16. If you are backing up to a tape drive (as specified in the **Destination** section of the **General** page), the **Unload the tape after backup** option is active. Clicking this option activates the **Rewind the tape before unloading** option.

17. SQL Server 2008 Enterprise and later supports [backup compression](#). By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by checking **Compress backup**, and you can prevent compression by checking **Do not compress backup**.

To view the current backup compression default

- [View and Configure the backup compression default Option \(SQL Server](#)



To back up a transaction log

1. Execute the BACKUP LOG statement to back up the transaction log, specifying the following:
 - The name of the database to which the transaction log that you want to back up belongs.
 - The backup device where the transaction log backup is written.

Example (Transact-SQL)

Important

This example uses the `AdventureWorks2012` database, which uses the simple recovery model. To permit log backups, before taking a full database backup, the database was set to use the full recovery model. For more information, see [View or Change the Recovery Model of a Database \(SQL Server\)](#).

This example creates a transaction log backup for the `AdventureWorks2012` database to the previously created named backup device, `MyAdvWorks_FullRM_log1`.

```
BACKUP LOG AdventureWorks2012  
    TO MyAdvWorks_FullRM_log1;
```

GO



Related Tasks

- [Restore a Transaction Log Backup \(SQL Server\)](#)
- [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#)
- [Troubleshoot a Full Transaction Log \(Error 9002\)](#)



See Also

[BACKUP](#)

[Working with Transaction Log Backups](#)

[Maintenance Plans](#)

[Using File Backups](#)

Tail-Log Backups

This topic is relevant only for backup and restore of SQL Server databases that are using the full or bulk-logged recovery models.

A *tail-log backup* captures any log records that have not yet been backed up (the *tail of the log*) to prevent work loss and to keep the log chain intact. Before you can recover a SQL Server database to its latest point in time, you must back up the tail of its transaction log. The tail-log backup will be the last backup of interest in the recovery plan for the database.



Note

Not all restore scenarios require a tail-log backup. You do not need a tail-log backup if the recovery point is contained in an earlier log backup. Also, a tail-log backup is unnecessary if you are moving or replacing (overwriting) a database and do not need to restore it to a point of time after its most recent backup.

In this Topic:

- Scenarios That Require a Tail-Log Backup
- Tail-Log Backups That Have Incomplete Backup Metadata
- Related Tasks
- Related Content

Scenarios That Require a Tail-Log Backup

We recommend that you take a tail-log backup in the following scenarios:

- If the database is online and you plan to perform a restore operation on the database, begin by backing up the tail of the log. To avoid an error for an online database, you must use the ... WITH NORECOVERY option of the [BACKUP](#) Transact-SQL statement.
- If a database is offline and fails to start and you need to restore the database, first back up the tail of the log. Because no transactions can occur at this time, using the WITH NORECOVERY is optional.
- If a database is damaged, try to take a tail-log backup by using the WITH CONTINUE_AFTER_ERROR option of the BACKUP statement.

On a damaged database backing up the tail of the log can succeed only if the log files are undamaged, the database is in a state that supports tail-log backups, and the database does not contain any bulk-logged changes. If a tail-log backup cannot be created, any transactions committed after the latest log backup are lost.

The following table summarizes the BACKUP NORECOVERY and CONTINUE_AFTER_ERROR options.

BACKUP LOG option	Comments
NORECOVERY	Use NORECOVERY whenever you intend to continue with a restore operation on the database. NORECOVERY takes the

BACKUP LOG option	Comments
	<p>database into the restoring state. This guarantees that the database does not change after the tail-log backup.</p> <p>The log is truncated unless the NO_TRUNCATE option or COPY_ONLY option is also specified.</p> <p> Important We recommend that you avoid using NO_TRUNCATE, except when the database is damaged.</p>
CONTINUE_AFTER_ERROR	<p>Use CONTINUE_AFTER_ERROR only if you are backing up the tail of a damaged database.</p> <p> Note When you use back up the tail of the log on a damaged database, some of the metadata ordinarily captured in log backups might be unavailable. For more information, see Tail-Log Backups That Have Incomplete Backup Metadata, later in this topic.</p>

Tail-Log Backups That Have Incomplete Backup Metadata

Tail log backups capture the tail of the log even if the database is offline, damaged, or missing data files. This might cause incomplete metadata from the restore information commands and **msdb**. However, only the metadata is incomplete; the captured log is complete and usable.

If a tail-log backup has incomplete metadata, in the [backupset](#) table, **has_incomplete_metadata** is set to **1**. Also, in the output of [RESTORE HEADERONLY](#), **HasIncompleteMetadata** is set to **1**.

If the metadata in a tail-log backup is incomplete, the [backupfilegroup](#) table will be missing most of the information about filegroups at the time of the tail-log backup. Most of the **backupfilegroup** table columns are NULL; the only meaningful columns are as follows:

- **backup_set_id**

- **filegroup_id**
- **type**
- **type_desc**
- **is_readonly**

Related Tasks

To create a tail-log backup, see [Back Up the Tail of the Transaction Log \(SQL Server Management Studio\)](#).

To restore a transaction log backup, see [Restore a Transaction Log Backup \(SQL Server Management Studio\)](#).

Related Content

None.

See Also

[BACKUP \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[Back Up and Restore of SQL Server Databases](#)

[Copy-Only Backups](#)

[Create Transaction Log Backups \(SQL Server\)](#)

[Apply Transaction Log Backups](#)

Back Up the Transaction Log When the Database Is Damaged

This topic describes how to back up a transaction log when the database is damaged in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Recommendations
 - Security
- **To back up the transaction log when the database is damaged, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The BACKUP statement is not allowed in an explicit or implicit transaction.

Recommendations

- For a database that uses either the full or bulk-logged recovery model, you generally need to back up the tail of the log before beginning to restore the database. You also should back up the tail of the log of the primary database before failing over a log shipping configuration. Restoring the tail-log backup as the final log backup before recovering the database avoids work loss after a failure. For more information about tail-log backups, see [Overview of Tail-Log Backups](#).

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

▶ To back up the tail of the transaction log

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Database** list box, verify the database name. You can optionally select a different database from the list.
5. Verify that the recovery model is either **FULL** or **BULK_LOGGED**.

6. In the **Backup type** list box, select **Transaction Log**.
7. Leave **Copy Only Backup** deselected.
8. In the **Backup set** area, either accept the default backup set name suggested in the **Name** text box, or enter a different name for the backup set.
9. In the **Description** text box, enter a description for the tail-log backup.
10. Specify when the backup set will expire:
 - To have the backup set expire after a specific number of days, click **After** (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.

The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (**Database Settings** page). To access this dialog box, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.
 - To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.
11. Choose the type of backup destination by clicking **Disk** or **Tape**. To select the paths of up to 64 disk or tape drives containing a single media set, click **Add**. The selected paths are displayed in the **Backup to** list box.

To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.
12. On the **Options** page, select an **Overwrite Media** option, by clicking one of the following:
 - **Back up to the existing media set**

For this option, click either **Append to the existing backup set** or **Overwrite all existing backup sets**.

Optionally, select **Check media set name and backup set expiration** to cause the backup operation to verify the date and time at which the media set and backup set expire.

Optionally, enter a name in the **Media set name** text box. If no name is specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see whether the actual name matches the name you enter here.

If you leave the media name blank and check the box to check it against the media, success will equal the media name on the media also being blank.
 - **Back up to a new media set, and erase all existing backup sets**

For this option, enter a name in the **New media set name** text box, and, optionally, describe the media set in the **New media set description** text box.

For more information about media set options, see [Media Sets, Media Families, and Backup Sets \(SQL Server\)](#).

13. In the **Reliability** section, optionally, check:

- **Verify backup when finished.**
- **Perform checksum before writing to media.**
- **Continue on checksum error**

For information on checksums, see [Detecting and Coping with Media Errors](#).

14. In the **Transaction log** section, check **Back up the tail of the log, and leave database in the restoring state.**

This is equivalent to specifying the following [BACKUP](#) statement:

```
BACKUP LOG <database_name> TO <backup_device> WITH NORECOVERY
```

Important

At restore time, the Restore Database dialog box displays the type of a tail-log backup as **Transaction Log (Copy Only)**.

15. If you are backing up to a tape drive (as specified in the **Destination** section of the **General** page), the **Unload the tape after backup** option is active. Clicking this option activates the **Rewind the tape before unloading** option.

16. SQL Server 2008 Enterprise and later supports [backup compression](#). By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by checking **Compress backup**, and you can prevent compression by checking **Do not compress backup**.

To view the current backup compression default

- [How to: View and Configure the backup compression default Option \(SQL Server Management Studio\)](#)



Using Transact-SQL

To create a backup of the currently active transaction log

1. Execute the BACKUP LOG statement to back up the currently active transaction log, specifying:
 - The name of the database to which the transaction log to back up belongs.
 - The backup device where the transaction log backup will be written.
 - The NO_TRUNCATE clause.

This clause allows the active part of the transaction log to be backed up even if the database is inaccessible, provided that the transaction log file is

accessible and undamaged.

Example (Transact-SQL)

Note

This example uses the `AdventureWorks2012`, which uses the simple recovery model. To permit log backups, before taking a full database backup, the database was set to use the full recovery model. For more information, see [View or Change the Recovery Model of a Database \(SQL Server\)](#).

This example backs up the currently active transaction log when a database is damaged and inaccessible, if the transaction log is undamaged and accessible.

```
BACKUP LOG AdventureWorks2012
    TO MyAdvWorks_FullRM_log1
    WITH NO_TRUNCATE;
```

GO



See Also

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#)

[Back Up Database \(Options Page\)](#)

[Back Up Database \(General Page\)](#)

[Apply Transaction Log Backups \(SQL Server\)](#)

[BACKUP \(Transact-SQL\)](#)

[Performing File Restores \(Simple Recovery Model\)](#)

[Performing File Restores \(Full Recovery Model\)](#)

Backup Devices

During a backup operation on a SQL Server database, the backed up data (the *backup*) is written to a physical backup device. This physical backup device is initialized when the first backup in a media set is written to it. The backups on a set of one or more backup devices compose a single media set.

In this Topic:

- Terms and Definitions
- Using Disk Backup Devices
- Using Tape Devices
- Using a Logical Backup Device

- Mirrored Backup Media Sets
- Archiving SQL Server Backups
- Related Tasks

Terms and Definitions

backup disk

A hard disk or other disk storage media that contains one or more backup files. A backup file is a regular operating system file.

media set

An ordered collection of backup media, tapes or disk files, that uses a fixed type and number of backup devices. For more information about media sets, see [Media Sets, Media Families, and Backup Sets](#).

physical backup device

Either a tape drive or a disk file that is provided by the operating system. A backup can be written to from 1 to 64 backup devices. If a backup requires multiple backup devices, the devices all must correspond to a single type of device (disk or tape).

Using Disk Backup Devices

In This Section:

- Specifying a Backup File by Using Its Physical Name (Transact-SQL)
- Specifying the Path of a Disk Backup File
- Backing Up to a File on a Network Share

If a disk file fills while a backup operation is appending a backup to the media set, the backup operation fails. The maximum size of a backup file is determined by the free disk space available on the disk device; therefore, the appropriate size for a backup disk device depends on the size of your backups.

A disk backup device could be a simple disk device, such as an ATA drive. Alternatively, you could use a hot-swappable disk drive that would let you transparently replace a full disk on the drive with an empty disk. A backup disk can be a local disk on the server or a remote disk that is a shared network resource. For information about how to use a remote disk, see [Backing Up to a File on a Network Share](#), later in this topic.

SQL Server management tools are very flexible at handling disk backup devices because they automatically generate a time-stamped name on the disk file.

Important

We recommend that a backup disk be a different disk than the database data and log disks. This is necessary to make sure that you can access the backups if the data or log disk fails.

Specifying a Backup File by Using Its Physical Name (Transact-SQL)

The basic [BACKUP](#) syntax for specifying a backup file by using its physical device name is:

```
BACKUP DATABASE database_name
```

```
    TO DISK = { 'physical_backup_device_name' | @physical_backup_device_name_var }
```

For example:

```
BACKUP DATABASE AdventureWorks2012
```

```
    TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak' ;
```

```
GO
```

To specify a physical disk device in a [RESTORE](#) statement, the basic syntax is:

```
RESTORE { DATABASE | LOG } database_name
```

```
    FROM DISK = { 'physical_backup_device_name' | @physical_backup_device_name_var }
```

For example,

```
RESTORE DATABASE AdventureWorks2012
```

```
    FROM DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak' ;
```

Specifying the Path of a Disk Backup File

When you are specifying a backup file, you should enter its full path and file name. If you specify only the file name or a relative path when you are backing up to a file, the backup file is put in the default backup directory. The default backup directory is C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Backup, where *n* is the number of the server instance. Therefore, for the default server instance, the default backup directory is: C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup.

To avoid ambiguity, especially in scripts, we recommend that you explicitly specify the path of the backup directory in every DISK clause. However, this is less important when you are using Query Editor. In that case, if you are sure that the backup file resides in the default backup directory, you can omit the path from a DISK clause. For example, the following BACKUP statement backs up the database to the default backup directory.

```
BACKUP DATABASE AdventureWorks2012
```

```
    TO DISK = 'AdventureWorks2012.bak' ;
```

```
GO
```



Note

The default location is stored in the **BackupDirectory** registry key under **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.n\MSSQLServer**.

Backing Up to a File on a Network Share

For SQL Server to access a remote disk file, the SQL Server service account must have access to the network share. This includes having the permissions needed for backup operations to write to the network share and for restore operations to read from it. The availability of network drives and permissions depends on the context in which SQL Server service is running:

- To back up to a network drive when SQL Server is running in a domain user account, the shared drive must be mapped as a network drive in the session where SQL Server is running. If you start `Sqlservr.exe` from command line, SQL Server sees any network drives you have mapped in your login session.
- When you run `Sqlservr.exe` as a service, SQL Server runs in a separate session that has no relation to your login session. The session in which a service runs can have its own mapped drives, although it usually does not.
- You can connect with the network service account by using the computer account instead of a domain user. To enable backups from specific computers to a shared drive, grant access to the computer accounts. As long as the `Sqlservr.exe` process that is writing the backup has access, it is irrelevant whether the user sending the `BACKUP` command has access.



Important

Backing up data over a network can be subject to network errors; therefore, we recommend that when you are using a remote disk you verify the backup operation after it finishes. For more information, see [RESTORE VERIFYONLY \(Transact-SQL\)](#).

Specifying a Universal Naming Convention (UNC) Name

To specify a network share in a backup or restore command, you should use the fully qualified universal naming convention (UNC) name of the file for the backup device. A UNC name has the form `\\Systemname\ShareName\Path\FileName`.

For example:

```
BACKUP DATABASE AdventureWorks2012
    TO DISK =
    '\\BackupSystem\BackupDisk1\AW_backups\AdventureWorksData.Bak' ;
GO
```

Using Tape Devices



Note

Support for tape backup devices will be removed in a future version of SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

In This Section:

- Specifying a Backup Tape by Using Its Physical Name (Transact-SQL)
- Tape-Specific BACKUP and RESTORE Options (Transact-SQL)
- Managing Open Tapes

Backing up SQL Server data to tape requires that the tape drive or drives be supported by the Microsoft Windows operating system. Additionally, for the given tape drive, we recommend that you use only tapes recommended by the drive manufacturer. For more information about how to install a tape drive, see the documentation for the Windows operating system.

When a tape drive is used, a backup operation may fill one tape and continue onto another tape. Each tape contains a media header. The first media used is called the *initial tape*. Each successive tape is known as a *continuation tape* and has a media sequence number that is one higher than the previous tape. For example, a media set associated with four tape devices contains at least four initial tapes (and, if the database does not fit, four series of continuation tapes). When appending a backup set, you must mount the last tape in the series. If the last tape is not mounted, the Database Engine scans forward to the end of the mounted tape and then requires that you change the tape. At that point, mount the last tape.

Tape backup devices are used like disk devices, with the following exceptions:

- The tape device must be connected physically to the computer that is running an instance of SQL Server. Backing up to remote tape devices is not supported.
- If a tape backup device is filled during the backup operation, but more data still must be written, SQL Server prompts for a new tape and continues the backup operation after a new tape is loaded.

Specifying a Backup Tape by Using Its Physical Name (Transact-SQL)

The basic [BACKUP](#) syntax for specifying a backup tape using the physical device name of the tape drive is:

```
BACKUP { DATABASE | LOG } database_name  
    TO TAPE = { 'physical_backup_device_name' | @physical_backup_device_name_var }
```

For example:

```
BACKUP LOG AdventureWorks2012  
    TO TAPE = '\\.\tape0';  
GO
```

To specify a physical tape device in a [RESTORE](#) statement, the basic syntax is:

```
RESTORE { DATABASE | LOG } database_name
```

```
FROM TAPE = { 'physical_backup_device_name' | @physical_backup_device_name_var }
```

Tape-Specific BACKUP and RESTORE Options (Transact-SQL)

To facilitate tape management, the BACKUP statement provides the following tape-specific options:

- { NOUNLOAD | UNLOAD }

You can control whether a backup tape is unloaded automatically from the tape drive after a backup or restore operation. UNLOAD/NOUNLOAD is a session setting that persists for the life of the session or until it is reset by specifying the alternative.

- { REWIND | NOREWIND }

You can control whether SQL Server keeps the tape remains open after the backup or restore operation or releases and rewinds the tape after it fills. The default behavior is to rewind the tape (REWIND).



Note

For more information about the BACKUP syntax and arguments, see [BACKUP \(Transact-SQL\)](#). For more information about the RESTORE syntax and arguments, see [RESTORE \(Transact-SQL\)](#) and [RESTORE Arguments \(Transact-SQL\)](#), respectively.

Managing Open Tapes

To view a list of open tape devices and the status of mount requests, query the [sys.dm_io_backup_tapes](#) dynamic management view. This view shows all the open tapes. These include in-use tapes that are temporarily idle while they wait for the next BACKUP or RESTORE operation.

If a tape has been accidentally left open, the fastest way to release the tape is by using the following command: RESTORE REWINDONLY FROM TAPE = *backup_device_name*. For more information, see [RESTORE REWINDONLY](#).

Using a Logical Backup Device

A *logical backup device* is an optional, user-defined name that points to a specific physical backup device (a disk file or tape drive). A logical backup device lets you use indirection when referencing the corresponding physical backup device.

Defining a logical backup device involves assigning a logical name to a physical device. For example, a logical device, AdventureWorksBackups, could be defined to point to the Z:\SQLServerBackups\AdventureWorks2012.bak file or the \\.\tape0 tape drive. Backup and restore commands can then specify AdventureWorksBackups as the backup device, instead of DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak' or TAPE = '\\.\tape0'.

The logical device name must be unique among all the logical backup devices on the server instance. To view the existing logical device names, query the [sys.backup_devices](#)

catalog view. This view displays the name of each logical backup device and describes the type and physical file name or path of the corresponding physical backup device. After a logical backup device is defined, in a BACKUP or RESTORE command, you can specify the logical backup device instead of the physical name of the device. For example, the following statement backs up the AdventureWorks2012 database to the AdventureWorksBackups logical backup device.

```
BACKUP DATABASE AdventureWorks2012  
    TO AdventureWorksBackups ;  
GO
```

Note

In a given BACKUP or RESTORE statement, the logical backup device name and the corresponding physical backup device name are interchangeable.

One advantage of using a logical backup device is that it is simpler to use than a long path. Using a logical backup device can help if you plan to write a series of backups to the same path or to a tape device. Logical backup devices are especially useful for identifying tape backup devices.

A backup script can be written to use a particular logical backup device. This lets you switch to a new physical backup devices without updating the script. Switching involves the following process:

1. Dropping the original logical backup device.
2. Defining a new logical backup device that uses the original logical device name but maps to a different physical backup device. Logical backup devices are especially useful for identifying tape backup devices.

Mirrored Backup Media Sets

Mirroring of backup media sets reduces the effect of backup-device malfunctions. These malfunctions are especially serious because backups are the last line of defense against data loss. As the sizes of databases grow, the probability increases that a failure of a backup device or media will make a backup nonrestorable. Mirroring backup media increases the reliability of backups by providing redundancy for the physical backup device. For more information, see [Using Mirrored Backup Media Sets](#).

Note

Mirrored backup media sets are supported only in SQL Server 2005 Enterprise Edition and later versions.

Archiving SQL Server Backups

We recommend that you use a file system backup utility to archive the disk backups and that you store the archives off-site. Using disk has the advantage that you use the network to write the archived backups onto an off-site disk. Using tape has the advantage of letting you accumulate a series of archived backups until you are sure that you no longer need them.

A common archiving approach is to write SQL Server backups onto a local backup disk, archive them to tape, and then store the tapes off-site.

Related Tasks

To specify a disk device (SQL Server Management Studio)

- [Specify a Disk or Tape as a Backup Destination \(SQL Server Management Studio\)](#)

To specify a tape device (SQL Server Management Studio)

- [Specify a Disk or Tape as a Backup Destination \(SQL Server Management Studio\)](#)

To define a logical backup device

- [sp_addumpDevice](#)
- [Define a Logical Backup Device for a Disk File \(SQL Server Management Studio\)](#)
- [Define a Logical Backup Device for a Tape Drive \(SQL Server Management Studio\)](#)
- `T:Microsoft.SqlServer.Management.Smo.BackupDevice` (SMO)

To use a logical backup device

- [Specify a Disk or Tape as a Backup Destination \(SQL Server Management Studio\)](#)
- [Restore a Backup From a Device \(SQL Server Management Studio\)](#)
- [BACKUP \(Transact-SQL\)](#)
- [RESTORE \(Transact-SQL\)](#)

To View Information About Backup Devices

- [View Information About Backups](#)
- [View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#)
- [View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)

To delete a logical backup device

- [sp_dropdevice](#)
- [Delete a Backup Device \(SQL Server Management Studio\)](#)



See Also

[SQL Server, Backup Device Object](#)

[BACKUP \(Transact-SQL\)](#)

[Maintenance Plans](#)

[Media Sets, Media Families, and Backup Sets](#)

[RESTORE \(Transact-SQL\)](#)

[RESTORE LABELONLY \(Transact-SQL\)](#)

[sys.backup_devices \(Transact-SQL\)](#)

[sys.dm_io_backup_tapes](#)

[Mirrored Backup Media Sets](#)

Define a Logical Backup Device for a Disk File

This topic describes how to define a logical backup device for a disk file in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. A logical device is a user-defined name that points to a specific physical backup device (a disk file or tape drive). The initialization of the physical device occurs later, when a backup is written to the backup device.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Recommendations
 - Security
- **To define a logical backup device for a disk file, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The logical device name must be unique among all the logical backup devices on the server instance. To view the existing logical device names, query the [sys.backup_devices](#) catalog view.

Recommendations

- We recommend that a backup disk be a different disk than the database data and log disks. This is necessary to make sure that you can access the backups if the data or log disk fails.

Security

Permissions

Requires membership in the **diskadmin** fixed server role.

Requires permission to write to the disk.



Using SQL Server Management Studio

▶ To define a logical backup device for a disk file

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Server Objects**, and right-click **Backup Devices**.
3. Click **New Backup Device**. The **Backup Device** dialog box opens.
4. Enter a device name.
5. For the destination, click **File** and specify the full path of the file.
6. To define the new device, click **OK**.

To back up to this new device, add it to the **Back up to:** field in the **Back up Database (General)** dialog box. For more information, see [How to: View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#).



Using Transact-SQL

▶ To define a logical backup for a disk file

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. This example shows how to use [sp_addumpdevice](#) to define a logical backup device for a disk file. The example adds the disk backup device named `mydiskdump`, with the physical name `c:\dump\dump1.bak`.

```
USE AdventureWorks2012 ;
```

```
GO
```

```
EXEC sp_addumpdevice 'disk', 'mydiskdump', 'c:\dump\dump1.bak' ;
```

```
GO
```



See Also

[BACKUP \(Transact-SQL\)](#)

[Backup Devices](#)

[sys.backup_devices \(Transact-SQL\)](#)

[sp_addumpdevice \(Transact-SQL\)](#)

[sp_dropdevice \(Transact-SQL\)](#)

[How to: Create a Tape Backup Device \(SQL Server Management Studio\)](#)

[How to: View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)

Define a Logical Backup Device for a Tape Drive

This topic describes how to define a logical backup device for a tape drive in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. A logical device is a user-defined name that points to a specific physical backup device (a disk file or tape drive). The initialization of the physical device occurs later, when a backup is written to the backup device.



Note

Support for tape backup devices will be removed in a future version of SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Security
- **To define a logical backup device for a tape drive, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The tape drive or drives must be supported by the Microsoft Windows operating system.
- The tape device must be connected physically to the computer that is running an instance of SQL Server. Backing up to remote tape devices is not supported.

Security

Permissions

Requires membership in the **diskadmin** fixed server role.

Requires permission to write to the disk.



Using SQL Server Management Studio

▶ To define a logical backup device for a tape drive

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Server Objects**, and then right-click **Backup Devices**.
3. Click **New Backup Device**, which opens the **Backup Device** dialog box.
4. Enter a device name.
5. For the destination, click **Tape** and select a tape drive that is not already associated with another backup device. If no such tape drives are available, the **Tape** option is inactive.
6. To define the new device, click **OK**.

To back up to this new device, add it to the **Back up to:** field in the **Back up Database (General)** dialog box. For more information, see [SQL Server Management Studio Tutorial](#).



Using Transact-SQL

▶ To define a logical backup device for a tape drive

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. This example shows how to use [sp_addumpdevice](#) to define a logical backup device for a tape. The example adds the tape backup device named `tapedump1`, with the physical name `\\.\tape0`.

```
USE AdventureWorks2012 ;
GO
EXEC sp_addumpdevice 'tape', 'tapedump1', '\\.\tape0' ;
GO
```



See Also

[BACKUP \(Transact-SQL\)](#)

[sys.backup_devices \(Transact-SQL\)](#)

[sp_addumpdevice \(Transact-SQL\)](#)

[sp_dropdevice \(Transact-SQL\)](#)

[Backup Devices](#)

[How to: Create a Disk Backup Device \(SQL Server Management Studio\)](#)

[How to: View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)

View the Contents of a Backup Tape or File

This topic describes how to view the content of a backup tape or file in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.



Note

Support for tape backup devices will be removed in a future version of SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

In This Topic

- **Before you begin:**
 - Security
- **To view the content of a backup tape or file, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

For information about security, see [RESTORE HEADERONLY \(Transact-SQL\)](#).

Permissions

In SQL Server 2008 and later versions, obtaining information about a backup set or backup device requires CREATE DATABASE permission. For more information, see [GRANT Database Permissions \(Transact-SQL\)](#).



Using SQL Server Management Studio

 **To view the content of a backup tape or file**

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database you want to backup, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Destination** section of the **General** page, click either **Disk** or **Tape**. In the **Back up to** list box, look for the disk file or tape you want.
If the disk file or tape is not displayed in the list-box, click **Add**. Select a file name or tape drive. To add it to the **Back up to** list-box, click **OK**.
5. In the **Back up to** list-box, select the path of the disk or tape drive you want to view, and click **Contents**. This opens the **Device Contents** dialog box.
6. The right-hand pane displays information about the media set and backup sets on the selected tape or file.



Using Transact-SQL

To view the content of a backup tape or file

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Use the [RESTORE HEADERONLY](#) statement. This example returns information about the file named AdventureWorks2012-FullBackup.bak.

```
USE AdventureWorks2012;  
RESTORE HEADERONLY  
FROM DISK = N'C:\AdventureWorks2012-FullBackup.bak' ;  
GO
```



See Also

[backupfilegroup \(Transact-SQL\)](#)

[backupfile \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[backupmediaset \(Transact-SQL\)](#)

[backupmediafamily \(Transact-SQL\)](#)

[Backup Devices](#)

[How to: Create a Disk Backup Device \(SQL Server Management Studio\)](#)

Specify a Disk or Tape As a Backup Destination

This topic describes how to specify a disk or tape as a backup destination in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

Note

Support for tape backup devices will be removed in a future version of SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

In This Topic

- **Before you begin:**
Security
- **To specify a disk or tape as a backup destination, using:**
SQL Server Management Studio
Transact-SQL

Before You Begin

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

To specify a disk or tape as a backup destination

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.

2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. In the **Destination** section of the **General** page, click **Disk** or **Tape**. To select the paths of up to 64 disk or tape drives containing a single media set, click **Add**.
To remove a backup destination, select it and click **Remove**. To view the contents of a backup destination, select it and click **Contents**.



Using Transact-SQL

▶ To specify a disk or tape as a backup destination

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. In the [BACKUP](#) statement, specify the file or device and its physical name. This example backs up the AdventureWorks2012 database to the disk file
Z:\SQLServerBackups\AdventureWorks2012.Bak.

```
USE AdventureWorks2012;
```

```
GO
```

```
BACKUP DATABASE AdventureWorks2012
```

```
TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.Bak'
```

```
GO
```



See Also

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Create a Disk Backup Device \(SQL Server Management Studio\)](#)

[How to: Create a Full Differential Backup \(SQL Server Management Studio\)](#)

[How to: Create a Tape Backup Device \(SQL Server Management Studio\)](#)

Device Contents

Use this dialog box to view the backup information. This information describes the device, the media, the media set, and the backup set or sets.

To use SQL Server Management Studio to view the contents of a backup device

- [View the Contents of a Backup Tape or File \(SQL Server\)](#)

- [View the Properties and Contents of a Logical Backup Device \(SQL Server\)](#)

Options

Media

A disk or set of tapes on which backup information is stored.

Media sequence

Lists the media sequence number, the family sequence number, and the mirror identifier, if any. The physical backup media are each tagged with a media sequence number that indicates the order in which the media were used. The initial backup medium is tagged with 1, the second (the first continuation tape) is tagged with 2, and so forth. When the backup set is restored, the media sequence numbers ensure that the operator that restores the backup mounts the correct media in the correct order.

Created on

Displays the media date.

Media Set

A media set is an ordered collection of backup media to which one or more backup operations have written using a constant number of backup devices.

Name

Displays the name of the media set.

Description

Displays the description media set.

Media family count

Displays the media family count. Each media set is a collection of one or more media families. All the output to a given single backup device (or group of mirrored backup devices) forms a single media family. Each media set contains one media family per separate device (or group of mirrored devices); for example, if a media set uses two nonmirrored backup devices, the media set contains two media families.

Backup sets

Displays information about the backup set or sets contained on the media. A backup set is the result of a successful backup operation whose content is distributed among the media on the set of backup devices.

Header	Values
Name	The name of the backup set.
Type	The type of backup performed: Full, Differential or Transaction Log.

Component	The backed-up component: Database, File, or <blank> (for transaction logs).
Server	The name of the instance of the Database Engine that performed the backup operation.
Database	The name of the database that was backed up.
Position	The position of the backup set in the volume.
Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.
Expiration	The date and time the backup set expires.

See Also

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

Backup Device (Media Contents Page)

Use the **Backup Device** dialog box to view the backup information. This information describes the device, the media, the media set, and the backup set or sets.

To use SQL Server Management Studio to view the contents of a backup device

- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)
- [How to: View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#)

Options

View information about the individual media, media set, and backup sets.

Media

A disk or set of tapes on which backup information is stored.

Media sequence

Lists the media sequence number, the family sequence number, and the mirror identifier, if any. The physical backup media are each tagged with a media sequence number that indicates the order in which the media were used. The initial backup media

is tagged with 1, the second (the first continuation tape) is tagged with 2, and so forth. When the backup set is restored, the media sequence numbers ensure that the operator restoring the backup mounts the correct media in the correct order.

Created on

Displays the creation date and time of the media set.

Media Set

A media set is an ordered collection of backup media to which one or more backup operations have written by using a constant number of backup devices.

Name

Displays the name of the media set, if any.

Description

Displays the description of the media set, if any.

Media family count

Displays the number of families in the media set. Each media set is a collection of one or more media families. All the output to a given single backup device (or group of mirrored backup devices) forms a single media family. Each media set contains one media family per separate device (or group of mirrored devices); for instance, if a media set uses two non-mirrored backup devices, the media set contains two media families.

Backup sets

Displays information about the backup set or sets contained on the media. A backup set is the result of a successful backup operation, whose content is distributed among the media on the set of backup devices.

Header	Values
Name	The name of the backup set.
Type	The type of backup performed: Full, Differential or Transaction Log.
Component	The backed-up component: Database, File, or <i><blank></i> (for transaction logs).
Server	The name of the instance of the Database Engine that performed the backup operation.
Database	The name of the database that was backed up.
Position	The position of the backup set in the

	volume.
Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.
Expiration	The date and time the backup set expires.



Related Tasks

- [Define a Logical Backup Device for a Disk File \(SQL Server\)](#)
- [Define a Logical Backup Device for a Tape Drive \(SQL Server\)](#)
- [Specify a Disk or Tape as a Backup Destination \(SQL Server\)](#)
- [Delete a Backup Device \(SQL Server\)](#)
- [Set the Expiration Date on a Backup \(SQL Server\)](#)
- [View the Contents of a Backup Tape or File \(SQL Server\)](#)
- [View the Data and Log Files In a Backup Set \(SQL Server\)](#)
- [View the Properties and Content of a Backup Device \(SQL Server\)](#)
- [Restore a Backup From a Device \(SQL Server\)](#)



See Also

[Backup Devices](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

Backup Device (General Page)

Use the **General** page to specify or view the general properties of a logical backup device.

To use SQL Server Management Studio to view the contents of a backup device

- [Working with Backup Media in SQL Server](#)
- [How to: View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#)

Options

Device name

View the name of an existing logical backup device or specify the name of a new logical

backup device.

Tape

View or select the destination tape device in the **Tape** list. This option is available only if a tape drive is attached to the computer that is running the instance of the Microsoft SQL Server Database Engine.



Note

Tape backup devices on remote computers are not valid backup destinations.

File

View the destination file of an existing logical backup device, or specify a destination file for a new logical backup device.

- For an existing logical backup device, the path of the backup file is displayed. The **File** field is not editable, and the Browse button is unavailable.
- For a new logical backup device, you must supply the path of the backup file for which you are defining the logical backup device. This file does not have to exist yet.

To specify a local backup file, you can click the Browse button to the right of the **File** text box. Then, in the **Locate Database Files** dialog box, you can navigate to any location on any of the fixed drives on the computer running the server instance. If the backup file does not exist yet, you must enter the filename you want to use in the **File name** field of that dialog box.

Alternatively, you can edit the **File** field manually to override the default path, file name, and extension. To specify a remote file as your backup destination, enter its fully qualified universal naming convention (UNC) name. For more information, see [Backup Devices](#).



Important

Backing up data over a network can be subject to network errors; therefore, we recommend that you verify the backup operation after it finishes. For more information, see [VERIFYONLY \(Transact-SQL\)](#).



Remarks

The backups on a set of one or more backup devices compose a single media set. A *media set* is an ordered collection of backup media, tapes or disk files, to which one or more backup operations have written using a fixed type and number of backup devices. For information about media sets, see [Media Sets, Media Families, and Backup Sets](#).

The physical backup device corresponding to a logical backup device is initialized when the first backup in the media set is written to the logical backup device. If the physical backup device is a file that does not exist yet, it is created at that time.



Related Tasks

- [Define a Logical Backup Device for a Disk File \(SQL Server\)](#)
- [Define a Logical Backup Device for a Tape Drive \(SQL Server\)](#)
- [Specify a Disk or Tape as a Backup Destination \(SQL Server\)](#)
- [Delete a Backup Device \(SQL Server\)](#)
- [Set the Expiration Date on a Backup \(SQL Server\)](#)
- [View the Contents of a Backup Tape or File \(SQL Server\)](#)
- [View the Data and Log Files In a Backup Set \(SQL Server\)](#)
- [View the Properties and Content of a Backup Device \(SQL Server\)](#)
- [Restore a Backup From a Device \(SQL Server\)](#)



See Also

[Backup Devices](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

Restore a Backup from a Device

This topic describes how to restore a backup from a device in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To restore a backup from a device, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore a backup from a device

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Restore**.
4. Click the type of restore operation you want (**Database, Files and Filegroups, or Transaction Log**). This opens the corresponding restore dialog box.
5. On the **General** page, in the **Restore source** section, click **From device**.
6. Click the browse button for the **From device** text box, which opens the **Specify Backup** dialog box.
7. In the **Backup media** text box, select **Backup Device**, and click the **Add** button to open the **Select Backup Device** dialog box.
8. In the **Backup device** text box, select the device you want to use for the restore operation.



Using Transact-SQL

▶ To restore a backup from a device

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. In the [RESTORE](#) statement, specify a logical or physical backup device to use for the backup operation. This example restores from a disk file that has the physical name Z:\SQLServerBackups\AdventureWorks2012.bak.

```
RESTORE DATABASE AdventureWorks2012
```

```
FROM DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak' ;
```



See Also

[RESTORE FILELISTONLY \(Transact-SQL\)](#)

[RESTORE HEADERONLY \(Transact-SQL\)](#)

[RESTORE LABELONLY \(Transact-SQL\)](#)

[RESTORE VERIFYONLY \(Transact-SQL\)](#)

[How to: Restore a Database Backup \(Transact-SQL\)](#)

[How to: Restore a Database Backup \(SQL Server Management Studio\)](#)

[How to: Restore a Differential Database Backup \(SQL Server Management Studio\)](#)

[How to: Create a New Database From an Existing Database Backup \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Create a Full Differential Backup \(SQL Server Management Studio\)](#)

Delete a Backup Device

This topic describes how to delete a backup device in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To delete a backup device, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

Permissions

Requires membership in the **diskadmin** fixed server role.



Using SQL Server Management Studio

 **To delete a backup device**

1. After connecting to the appropriate instance of the SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Server Objects**, and then expand **Backup Devices**.
3. Right-click the device you want, and then click **Delete**.
4. In the **Delete Object** dialog box, verify that the correct device name appears in the **Object Name** column.
5. Click **OK**.



Using Transact-SQL

▶ To delete a backup device

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query. This example shows how to use [sp_dropdevice](#) to delete a backup device. Execute the first example to create the `mybackupdisk` backup device and the physical name `c:\backup\backup1.bak`. Execute **sp_dropdevice** to delete the `mybackupdisk` backup device. The `delfile` parameter deletes the physical name.

```
--Define a backup device and physical name.
USE AdventureWorks2012 ;
GO
EXEC sp_addumpdevice 'disk', 'mybackupdisk', 'c:\backup\backup1.bak' ;
GO
--Delete the backup device and the physical name.
USE AdventureWorks2012 ;
GO
EXEC sp_dropdevice ' mybackupdisk ', 'delfile' ;
GO
```



See Also

[How to: View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)

[sys.backup_devices \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

Media Sets, Media Families, and Backup Sets

This topic introduces the basic backup-media terminology of SQL Server backup and restore and is intended for readers who are new to SQL Server. This topic describes the format that SQL Server uses for backup media, the correspondence between backup media and backup devices, the organization of backups on backup media, and several considerations for media sets and media families. The topic also describes the steps initializing or formatting backup media before you use it for the first time or replace an old media set with a new media set, how to overwrite old backup sets in a media set, and how to append new backup sets to a media set.

In this Topic:

- Terms and Definitions
- Overview of Media Sets, Media Families, and Backup Sets
- Using Media Sets and Families
- Related Tasks

Terms and Definitions

media set

An ordered collection of backup media, tapes or disk files, to which one or more backup operations have written using a fixed type and number of backup devices.

media family

Backups created on a single nonmirrored device or a set of mirrored devices in a media set

backup set

The backup content that is added to a media set by a successful backup operation.

Overview of Media Sets, Media Families, and Backup Sets

The backups on a set of one or more backup media compose a single media set. A *media set* is an ordered collection of *backup media*, tapes or disk files, to which one or more backup operations have written using a fixed type and number of backup devices. A given media set uses either tape drives or disk drives, but not both. For example, the backup devices associated with a media set might be three tape drives named `\\.\TAPE0`, `\\.\TAPE1`, and `\\.\TAPE2`. That media set contains only tapes, starting with a minimum of three tapes (one per drive). The type and number of backup devices

are established when a media set is created, and they cannot be changed. However, if necessary, between backup and restore operations a given device can be replaced with a device of the same type.

A media set is created on the backup media during a backup operation by formatting the backup media. For more information, see [Creating a New Media Set](#), later in this topic. After formatting, each file or tape contains a media header for the media set and is ready to receive backup content. With the header in place, the backup operation proceeds to back up the specified data to the backup media on all of the backup devices specified for the operation.

Note

Media sets can be mirrored to protect against a damaged media volume (a tape or disk file). For more information, see [Using Mirrored Backup Media Sets](#).

SQL Server 2008 Enterprise and later supports compressing backups. Compressed and uncompressed backups cannot occur together in a media set. Any edition of SQL Server 2008 or later can read compressed backups. For more information, see [Backup Compression \(SQL Server\)](#).

Media Families

Backups created on a single nonmirrored device or a set of mirrored devices in a media set constitute a *media family*. The number of backup devices used for the media set determines the number of media families in a media set. For example, if a media set uses two nonmirrored backup devices, the media set contains two media families.

Note

In a mirrored media set, each media family is mirrored. For example, if six backup devices are used to format a media set, where two mirrors are used, there are three media families, each containing two equivalent copies of backup data. For more information about mirrored media sets, see [Using Mirrored Backup Media Sets](#).

Each tape or disk in a media family is assigned a *media sequence number*. The media sequence number of a disk is always 1. In a tape media family, the sequence number of the initial tape is 1, the sequence number of the second tape is 2, and so forth. For more information, see [Using Media Sets and Families](#).

The Media Header

Every volume of backup media (disk file or tape) contains a media header that is created when by the first backup operation that uses the tape (or disk). That header remains intact until the media is reformatted.

The media header contains all of the information required to identify the media (disk file or tape) and its place within the media family to which it belongs. This information includes:

- The name of the media.
The media name is optional, but we recommend consistently using media names that clearly identify your media. A media name is assigned by whoever formats the media.
- The unique identification number of the media set.
- The number of media families in the media set.
- The sequence number of the media family containing this media.
- The unique identification number for the media family.
- The sequence number of this media in the media family. For a disk file, this value is always 1.
- Whether the media description contains an MTF media label or a media description.



Note

All media that is used for a backup or restore operation use a standard backup format called Microsoft Tape Format (MTF). MTF allows users to specify a tape label that contains a MTF-specific description. SQL Server preserves any MTF media label written by another application but does not write MTF media labels.

- The Microsoft Tape Format media label or the media description (in free-form text).
- The name of the backup software that wrote the label.
- The unique vendor identification number of the software vendor that formatted the media.
- The date and time the label was written.
- The number of mirrors in the set (1-4); 1 indicates an unmirrored device.

SQL Server 2012 can process media formatted by earlier versions of SQL Server.

Backup Sets

A successful backup operation adds a single *backup set* to the media set. The backup set is described in terms of the media set to which the backup belongs. If the backup media consists of only one media family, that family contains the entire backup set. If the backup media consists of multiple media families, the backup set is distributed among them. On each medium, the backup set contains a header that describes the backup set.

The following example shows a Transact-SQL statement that creates a media set called `MyAdvWorks_MediaSet_1` for the `AdventureWorks2012` database using three tape drives as backup devices:

```
BACKUP DATABASE AdventureWorks2012
```

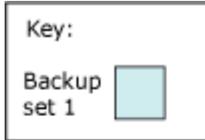
```
TO TAPE = '\\.\tape0', TAPE = '\\.\tape1', TAPE = '\\.\tape2'
```

```
WITH
```

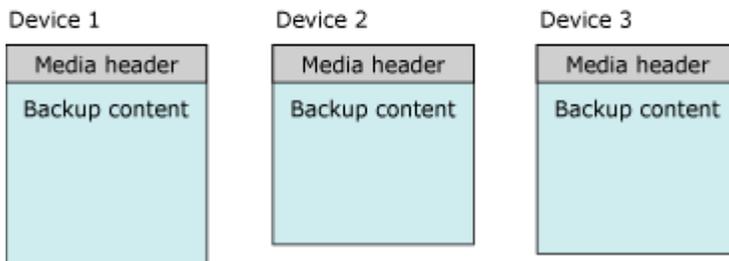
```
FORMAT,
```

```
MEDIANAME = 'MyAdvWorks_MediaSet_1'
```

If successful, this backup operation results in a new media set containing a new media header and one backup set spread across three tapes. The following figure illustrates these results:



Note: Each device holds approximately 1/3 of each backup set.



Typically, after a media set is created, subsequent backup operations, one after another, append their backup sets to the media set. All of the media used by a backup set make up the media set, regardless of the number of media or backup devices involved. Backup sets are sequentially numbered by their position in the media set, allowing you to specify which backup set to restore.

Every backup operation to a media set must write to the same number and type of backup devices. With multiple devices, as with the first backup set, the content of every subsequent backup set is distributed among the backup media on all of the devices. To continue the above example, a second backup operation (a differential backup) appends information to the same media set:

```
BACKUP DATABASE AdventureWorks2012
```

```
TO TAPE = '\\.\tape0', TAPE = '\\.\tape1', TAPE = '\\.\tape2'
```

```
WITH
```

```
NOINIT,
```

```
MEDIANAME = 'AdventureWorksMediaSet1',
```

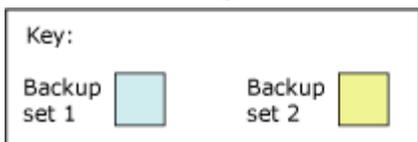
```
DIFFERENTIAL
```



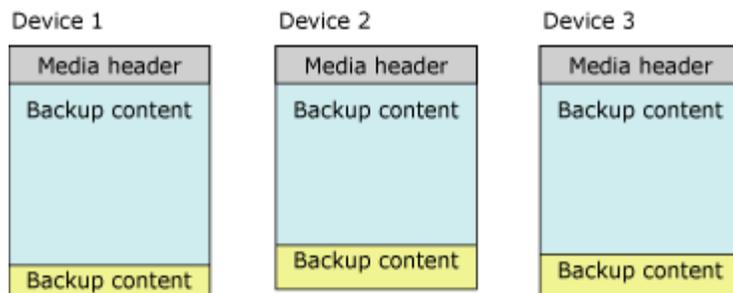
Note

The NOINIT option is the default, but is included for clarity.

If the second backup operation succeeds, it writes a second backup set to the media set, with the following distribution of backup content:



Note: Each device holds approximately 1/3 of each backup set.



When you are restoring backups, you can use you the FILE option to specify which backups you want to use. The following example shows the use of FILE = *backup_set_file_number* clauses when restoring a full database backup of the database followed by a differential database backup on the same media set. The media set uses three backup tapes, which are on tape drives \\.\tape0, tape1, and tape2.

```
RESTORE DATABASE AdventureWorks2012 FROM TAPE = '\\.\tape0', TAPE =
 '\\.\tape1', TAPE = '\\.\tape2'
WITH
MEDIANAME = 'AdventureWorksMediaSet1',
FILE=1,
NORECOVERY;
RESTORE DATABASE AdventureWorks2012 FROM TAPE = '\\.\tape0', TAPE =
 '\\.\tape1', TAPE = '\\.\tape2'
WITH
MEDIANAME = 'AdventureWorksMediaSet1',
FILE=2,
RECOVERY;
GO
```

For information about the history tables that store information about media sets and their media families and backup sets, see [Viewing Information About Backups](#).

The number of backup media in a media set depends on several factors:

- Number of backup devices
- Type of backup devices
- Number of backup sets

Using Media Sets and Families

This section discusses several considerations for using media sets and media families.

In This Section

- Creating a New Media Set
- Backing Up to an Existing Media Set
- Sequence Numbers
- Using Multiple Devices

Creating a New Media Set

To create a new media set, you must format the backup media (one or more tapes or disk files). The formatting process changes the backup media as follows:

1. Deletes the old header (if any), effectively deleting the previous contents of the backup media.

Formatting a tape device deletes all previous contents of the currently mounted tape. Formatting a disk affects only the file that you specify for the backup operation

2. Writes a new media header on the backup media (tape or disk file) on each of the backup devices.

Backing Up to an Existing Media Set

When you are backing up to an existing media set, you have the following two options:

- Append to the existing backup set.

To make the best possible use of the available space, new backup sets typically are appended to existing media set. Appending to the backup preserves any prior backups. For more information, see *Appending to Existing Backup Sets*, later in this section.



Note

Appending, which is the default behavior of the BACKUP, can be explicitly specified by using the NOINIT option.

- Overwrite all existing backup sets with the current backup, leaving the current media header in place.

SQL Server backup has safeguards to prevent you from accidentally overwriting media. However, backup can automatically overwrite backup sets that have reached a predefined expiration date.

For tape headers, leaving the header in place can make sense. For more information, see [Overwriting Backup Sets](#), later in this section.



Note

Overwriting existing backup sets is specified by using the INIT option of the BACKUP statement.

Appending to Existing Backup Sets

Backups performed at different times from the same or different databases can be stored on the same media. By appending another backup set to existing media, the previous contents of the media remain intact, and the new backup is written after the end of the last backup on the media.

By default, SQL Server always appends new backups to media. Appending can occur only at the end of the media. For example, if a media volume contains five backup sets, it is not possible to skip the first three backup sets to overwrite the fourth backup set with a new backup set.

If you use BACKUP WITH NOREWIND for a tape backup, the tape will be left open at the end of the operation. This allows you to append further backups to the tape without rewinding the tape and then scanning forward again to find the last backup set. You can find the list of open tape drives in the **sys.dm_io_backup_tapes** dynamic management view; for more information, see [BACKUP \(Transact-SQL\)](#).

Microsoft Windows backups and SQL Server backups can share the same media, but they are not interoperable. SQL Server backup cannot back up Windows data.



Important

SQL Server 2008 Enterprise and later supports compressing backups. Compressed and uncompressed backups cannot occur together in a media set. Any edition of SQL Server 2008 or later versions can read compressed backups. For more information, see [Backup Compression \(SQL Server\)](#).

Overwriting Backup Sets

Overwriting of existing backup sets is specified by using the INIT option of the BACKUP statement. This option overwrites all the backup sets on the media and preserve the media header, if any. If no media header exists, one is created.

For tape headers, leaving the header in place can make sense. For disk backup media, only the files used by the backup devices specified in the backup operation are overwritten; other files on the disk are unaffected. When overwriting backups, any existing media header is preserved, and the new backup is created as the first backup on

the backup device. If there is no existing media header, a valid media header with an associated media name and media description is written automatically. If the existing media header is invalid, the backup operation terminates. If the media is empty, the new media header is generated with the given `MEDIANAME`, `MEDIAPASSWORD`, and `MEDIADESCRIPTION`, if any.

noteDXDOC112778PADS **Security Note**

Beginning with SQL Server 2012, the `MEDIAPASSWORD` option is discontinued for creating backups. However, you can still restore backups created with passwords.

Backup media is not overwritten if either of the following conditions exists:

- The existing backups on the media have not expired. (If `SKIP` is specified, expiration is not checked.)

The expiration date specifies the date that the backup expires and can be overwritten by another backup. You can specify the expiration date when a backup is created. By default, the expiration date is determined by the **media retention** option set with **`sp_configure`**. For more information, see [sp_configure \(Transact-SQL\)](#).

- The media name, if provided, does not match the name on the backup media.

The media name is a descriptive name used for easy identification of the media.

If you are sure you want to overwrite the existing media (for example, if you know that the backups on the tape are no longer needed), you can explicitly skip these checks.

If the backup media is password protected by Microsoft Windows, Microsoft SQL Server does not write to the media. To overwrite media that is password protected, you must reinitialize the media.

Sequence Numbers

The correct order is important for multiple media families within a media set or multiple backup media within a media family. Therefore, backup assigns sequence numbers in the following ways:

- Sequential media families within a media set

Within a media set, the media families are numbered sequentially according to their position in the media set. The media-family number is recorded in the **`family_sequence_number`** column of the **`backupmediafamily`** table.

- Physical media within a media family

A media sequence number indicates the order of the physical media within a media family. The sequence number is 1 for the initial backup media. This is tagged with 1; the second (the first continuation tape) is tagged with 2; and so on. When the backup set is restored, the media sequence numbers make sure that the operator restoring the backup mounts the correct media in the correct order.

Multiple Devices

When you use multiple tape drives or disk files, the following considerations apply:

- For backup:
The complete media set that is created by a backup operation must be used by all subsequent backup operations. For example, if a media set was created by using two tape backup devices, all subsequent backup operations that involve the same media set must use two backup devices.
- For restore:
For any restore from disk backups and for any online restore, all the all media families must be concurrently mounted. For an offline restore from tape backups, you can process the media families from fewer backup devices. Each media family must be processed completely before starting to process another media family. Media families are always processed in parallel, unless they are being restored with a single device.

Related Tasks

To create a new media set

- [Back Up a Database \(SQL Server Management Studio\)](#) (**Back up to a new media set, and erase all existing backup sets** option)
- [BACKUP \(Transact-SQL\)](#) (FORMAT option)
- `P:Microsoft.SqlServer.Management.Smo.Backup.FormatMedia`

To append a new backup to existing media

- [Back Up a Database \(SQL Server Management Studio\)](#) (**Append to the existing backup set** option)
- [BACKUP \(Transact-SQL\)](#) (NOINIT option)
- [Backup.SqlBackup Method](#)

To overwrite existing backup sets

- [Back Up a Database \(SQL Server Management Studio\)](#) (**Overwrite all existing backup sets** option)
- [BACKUP \(Transact-SQL\)](#) (INIT option)
- [Backup.Initialize Property](#)

To set the expiration date

- [Set the Expiration Date on a Backup \(SQL Server Management Studio\)](#)

To view the media sequence and family sequence numbers

- [View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)
- [backupmediafamily \(Transact-SQL\)](#) (**family_sequence_number** column)

To view the backup sets on a particular backup device

- [View the Data and Log Files In a Backup Set \(SQL Server Management Studio\)](#)
- [View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#)
- [RESTORE HEADERONLY \(Transact-SQL\)](#)

To read the media header of the media on a backup device

- [RESTORE LABELONLY \(Transact-SQL\)](#)

See Also

[Backing Up and Restoring Databases in SQL Server](#)

[Possible Media Errors During Backup and Restore \(SQL Server\)](#)

[Backup History and Header Information](#)

[Mirrored Backup Media Sets](#)

[BACKUP \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[RESTORE REWINDONLY \(Transact-SQL\)](#)

[sp_configure](#)

Set the Expiration Date on a Backup

This topic describes how to set the expiration date on a backup in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To set the expiration date on a backup, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

Permissions

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions. However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.



Using SQL Server Management Studio

▶ To set the expiration date on a backup

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Back Up**. The **Back Up Database** dialog box appears.
4. On the **General** page, for **Backup set will expire**, specify an expiration date to indicate when the backup set can be overwritten by another backup:
 - To have the backup set expire after a specific number of days, click **After** (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.
The default value is set in the **Default backup media retention (in days)** option of the **Server Properties** dialog box (**Database Settings** page). To access this, right-click the server name in Object Explorer and select properties; then select the **Database Settings** page.
 - To have the backup set expire on a specific date, click **On**, and enter the date on which the set will expire.



Using Transact-SQL

▶ To set the expiration date on a backup

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. In the [BACKUP](#) statement, specify either the EXPIREDATE or RETAIN_DAYS option to determine when the SQL Server Database Engine can overwrite the backup. If

neither option is specified, the expiration date is determined by the [media retention](#) server configuration setting. This example uses the `EXPIREDATE` option to specify an expiration date of June 30, 2015 (6/30/2015).

```
USE AdventureWorks2012;  
GO  
BACKUP DATABASE AdventureWorks2012  
    TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.Bak'  
    WITH EXPIREDATE = '6/30/2015' ;  
GO
```



See Also

[How to: Back Up a Database \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Create a Full Differential Backup \(SQL Server Management Studio\)](#)

View the Data and Log Files in a Backup Set

This topic describes how to view the data and log files in a backup set in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To view the data and log files in a backup set, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

For information about security, see [RESTORE FILELISTONLY \(Transact-SQL\)](#)

Permissions

In SQL Server 2008 and later versions, obtaining information about a backup set or backup device requires `CREATE DATABASE` permission. For more information, see [GRANT Database Permissions \(Transact-SQL\)](#).



Using SQL Server Management Studio

▶ To view the data and log files in a backup set

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, and then click **Properties**, which opens the **Database Properties** dialog box.
4. In the **Select a Page** pane, click **Files**.
5. Look in the **Database files** grid for a list of the data and log files and their properties.



Using Transact-SQL

▶ To view the data and log files in a backup set

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Use the [RESTORE FILELISTONLY](#) statement. This example returns information about the second backup set (`FILE=2`) on the `AdventureWorksBackups` backup device.

```
USE AdventureWorks2012 ;  
RESTORE FILELISTONLY FROM AdventureWorksBackups  
    WITH FILE=2;  
GO
```



See Also

[backupfilegroup \(Transact-SQL\)](#)

[backupfile \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[backupmediaset \(Transact-SQL\)](#)

[backupmediafamily \(Transact-SQL\)](#)

[Backup Devices \(SQL Server\)](#)

Mirrored Backup Media Sets



Note

Mirrored backup media sets are supported only in SQL Server 2005 Enterprise Edition and later versions.

Mirroring a media set increases backup reliability by reducing the impact of backup-device malfunctions. These malfunctions are very serious because backups are the last line of defense against data loss. As databases grow, the probability increases that a failure of a backup device or media will make a backup nonrestorable. Mirroring backup media increases the reliability of backups by providing redundancy.



Note

For information about media sets in general, see [Media Sets, Media Families, and Backup Sets](#).

In this Topic:

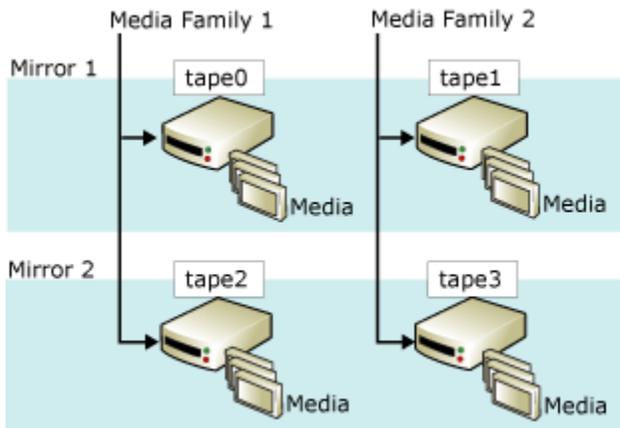
- Overview of Mirrored Media Sets
- Hardware Requirements for Backup Mirrors
- Related Tasks

Overview of Mirrored Media Sets

Media mirroring is a property of the media set. A *mirrored media set* consists of multiple copies (*mirrors*) of the media set. A media set contains one or more media families, each of which corresponds to a backup device. For example, if the TO clause of a BACKUP DATABASE statement lists three devices, BACKUP spreads the data among three media families, one per device. The number of media families and mirrors is defined when the media set is created (by a BACKUP DATABASE statement that specifies WITH FORMAT).

A mirrored media set possesses from two to four mirrors. Each mirror contains all the media families in the media set. The mirrors require the same number of devices, one per media family. Each mirror requires a separate backup device for each media family. For example, a mirrored media set that consists of four media families with three mirrors requires twelve backup devices. All of these devices must be equivalent. For example, tape drives that have the same model number from the same manufacturer.

The following illustration shows an example of a mirrored media set that consists of two media families with two mirrors. Each media family contains three media volumes, which are backed up one time per mirror.



Corresponding volumes on the mirrors have identical contents. This makes them interchangeable at restore time. For example, in the previous illustration, the third volume of tape2 is interchangeable with the third volume of tape0.

The SQL Server Database Engine guarantees that the mirrored media have identical contents by synchronizing writes to the devices. When any one of the mirrors fills, all the mirrors are spanned at one time.

Important

A mirrored media set cannot be implicitly broken (split) by removing a mirror. If any tape or disk in a mirror is damaged or reformatted, the mirror is no longer usable for additional backups. If at least one full mirror remains intact, the media set can be read. If every mirror loses a given media family, the media set is useless.

Backup and restore operations impose different requirements on whether all the mirrors must be present. For a backup operation to write (that is, to create or extend) a mirrored media set, all the mirrors must be present. In contrast, when you are restoring a backup from a mirrored media set, you can specify only a single mirror for each media family. You can restore from fewer devices than families, but each media family is processed only one time. In the presence of errors, however, having the other mirrors enables some restore problems to be resolved quickly. You can substitute a damaged media volume with the corresponding volume from another mirror. This is because RESTORE and RESTORE VERIFYONLY support substitution of damaged media with the corresponding backup-media volume from another mirror.

Hardware Requirements for Backup Mirrors

Mirroring applies both to disk and tape (disks do not support continuation tapes). As in earlier versions of SQL Server, all backup devices for a single backup or restore operation must be of the same type, disk or tape.

Within these broader classes, you must use similar devices that have the same properties. Insufficiently similar devices generate an error message (3212). To avoid the risk of a device mismatch, use devices that are equivalent, such as, only drives with the same model number from the same manufacturer.

Related Tasks

To back up to mirrored backup devices

- [Back Up to a Mirrored Media Set \(Transact-SQL\)](#)

See Also

[Detecting and Coping with Media Errors](#)

[RESTORE VERIFYONLY \(Transact-SQL\)](#)

[Backup Devices \(SQL Server\)](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

Back Up to a Mirrored Media Set (Transact-SQL)

This topic describes how to use the Transact-SQL [BACKUP](#) statement to specify a mirrored media set when backing up a SQL Server database. In your BACKUP statement, specify the first mirror in the TO clause. Then, specify each mirror in its own MIRROR TO clause. The TO and MIRROR TO clauses must specify the same number and type of backup devices.

Example

The following example creates the mirrored media set illustrated in the previous illustration and backs up the `AdventureWorks2012` database to both mirrors.

```
BACKUP DATABASE AdventureWorks2012
TO TAPE = '\\.\tape0', TAPE = '\\.\tape1'
MIRROR TO TAPE = '\\.\tape2', TAPE = '\\.\tape3'
WITH
    FORMAT,
    MEDIANAME = 'AdventureWorks2012Set1';
GO
```

Related Tasks

To restore from a mirrored backup

- [RESTORE \(Transact-SQL\)](#)

See Also

[BACKUP \(Transact-SQL\)](#)

[Mirrored Backup Media Sets \(SQL Server\)](#)

Backup History and Header Information

A complete history of all SQL Server backup and restore operations on a server instance is stored in the **msdb** database. This topic introduces the backup and restore history tables and also the Transact-SQL statements that are used to access backup history. The topic also discusses when listing database and transaction log files is useful and when to use media-header information compared to when to use backup-header information.

Important

To manage the risk of losing recent changes to your backup and restore history, back up **msdb** frequently. For information about which of the system databases you must back up, see [Backing Up and Restoring System Databases](#).

In This Topic:

- Backup and Restore History Tables
- Transact-SQL Statements for Accessing Backup History
- Database and Transaction Log Files
- Media-Header Information
- Backup-Header Information
- Comparison of Media-Header and Backup-Header Information
- Backup Verification
- Related Tasks

Backup and Restore History Tables

This section introduces the history tables that store backup and restore metadata in the **msdb** system database.

History table	Description
backupfile	Contains one row for each data or log file that is backed up.
backupfilegroup	Contains a row for each filegroup in a backup set.
backupmediafamily	Contains one row for each media family. If a media family resides in a mirrored media

History table	Description
	set, the family has a separate row for each mirror in the media set.
backupmediaset	Contains one row for each backup media set.
backupset	Contains a row for each backup set.
restorefile	Contains one row for each restored file. This includes files restored indirectly by filegroup name.
restorefilegroup	Contains one row for each restored filegroup.
restorehistory	Contains one row for each restore operation.



Note

When a restore is performed, backup history tables and restore history tables are modified.



Transact-SQL Statements for Accessing Backup History

The restore information statements correspond with information stored in certain backup history tables.

noteDXDOC112778PADS Security Note

In previous versions of SQL Server, any user could obtain information about backup sets and backup devices by using the RESTORE FILELISTONLY, RESTORE HEADERONLY, RESTORE LABELONLY, and RESTORE VERIFYONLY Transact-SQL statements. Because they reveal information about the content of the backup files, beginning in SQL Server 2008, these statements require CREATE DATABASE permission. This requirement secures your backup files and protects your backup information more fully than in previous versions. For information about this permission, see [GRANT Database Permissions \(Transact-SQL\)](#).

Information statement	Backup history table	Description
RESTORE FILELISTONLY	backupfile	Returns a result set that has a list of the database and log files that are contained

Information statement	Backup history table	Description
		in the specified backup set. For more information, see "Listing Database and Transaction Log Files," later in this topic.
RESTORE HEADERONLY	backupset	Retrieves all the backup header information for all backup sets on a particular backup device. The result from executing RESTORE HEADERONLY is a result set. For more information, see "Viewing the Backup-Header Information," later in this topic.
RESTORE LABELONLY	backupmediaset	Returns a result set that contains information about the backup media on a specified backup device. For more information, see "Viewing the Media-Header Information," later in this topic.

Column-Naming Conventions

For historical reasons, two different naming conventions exist. Old columns retain their original names. However, columns in SQL Server 2005 or later versions follow the naming conventions shown in the following table.

Context	Description
Columns returned by information commands	<i>WordWordWord</i> Example: DifferentialBaseLSN
Columns in msdb and in the catalog views	<i>word_word_word</i> Example: differential_base_lsn



Database and Transaction Log Files

Information that is displayed when the database and transaction log files are listed in a backup includes the logical name, physical name, file type (database or log), filegroup membership, file size (in bytes), the maximum allowed file size, and the predefined file growth size (in bytes). This information is useful, in the following situations, to determine the names of the files in a database backup before you restore the database backup:

- You have lost a disk drive that contains one or more of the files for a database.
You can list the files in the database backup to determine which files were affected, and then restore those files onto a different drive when you restore the whole database; or restore just those files and apply any transaction log backups created since the database was backed up.
- You are restoring a database from one server onto another server, but the directory structure and drive mapping does not exist on the server.
Listing the files in the backup let you determine which files are affected. For example, the backup contains a file that it has to restore to drive E, but the destination server does not have a drive E. The file must be relocated to another location, such as drive Z, when the file is restored.



Media-Header Information

Viewing the media header displays information about the media itself, instead of about the backups on the media. Media header information that is displayed includes the media name, description, name of the software that created the media header, and the date the media header was written.

Note

Viewing the media header is quick.

For more information, see [Comparison of Media-Header and Backup-Header Information](#), later in this topic.



Backup-Header Information

Viewing the backup header displays information about all SQL Server and non-SQL Server backup sets on the media. Information that is displayed includes the types of backup devices that are used, the types of backup (for example, database, transaction, file, or differential database), and backup start and stop date/time information. This information is useful when you have to determine which backup set on the tape to restore, or the backups that are contained on the media.

 **Note**

Viewing backup header information can take a long time for high-capacity tapes, because the whole media must be scanned to display information about each backup on the media.

For more information, see Comparison of Media-Header and Backup-Header Information, later in this topic.

Which Backup Set to Restore

You can use information in the backup header to identify which backup set to restore. The Database Engine numbers each backup set on the backup media. This lets you identify the backup set you want to restore by using its position on the media. For example, the following media contains three backup sets.

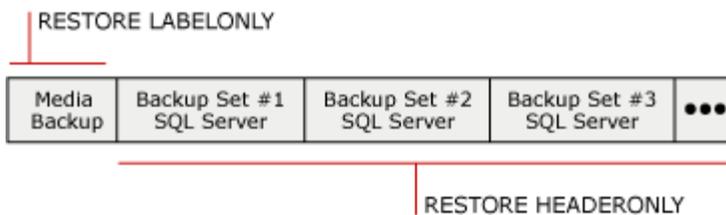
Media Backup	Backup Set #1 SQL Server	Backup Set #2 SQL Server	Backup Set #3 SQL Server	...
--------------	-----------------------------	-----------------------------	-----------------------------	-----

To restore a specific backup set, specify the position number of the backup set you want to restore. For example, to restore the second backup set, specify 2 as the backup set to restore.



Comparison of Media-Header and Backup-Header Information

The following illustration provides an example of the differences between viewing backup-header and media-header information. Obtaining the media header requires retrieving information from only the start of the tape. Obtaining the backup header requires scanning the whole tape to look at the header of every backup set.



 **Note**

When you use media sets that have multiple media families, the media header and backup set are written to all media families. Therefore, you only have to provide a single media family for these reporting operations.

For information about how to view the media-header, see "Viewing the Media-Header Information," earlier in this topic.

For information about how to view the backup header information for all backup sets on a backup device, see "Viewing the Backup-Header Information," earlier in this topic.



Backup Verification

Although not required, verifying a backup is a useful practice. Verifying a backup checks that the backup is intact physically, to ensure that all the files in the backup are readable and can be restored, and that you can restore your backup in the event you need to use it. It is important to understand that verifying a backup does not verify the structure of the data on the backup. However, if the backup was created using WITH CHECKSUMS, verifying the backup using WITH CHECKSUMS can provide a good indication of the reliability of the data on the backup.



Related Tasks

To delete old rows from backup and restore history tables

- [sp_delete_backuphistory](#)

To delete all rows for a specific database from backup and restore history tables

- [sp_delete_database_backuphistory](#)

To view the data and log files in a backup set

- [RESTORE FILELISTONLY \(Transact-SQL\)](#)

-

M:Microsoft.SqlServer.Management.Smo.Restore.ReadFileList(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To view media header information

- [RESTORE LABELONLY \(Transact-SQL\)](#)

- [How to: View the Properties and Contents of a Backup Device \(SQL Server Management Studio\)](#)

- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)

-

M:Microsoft.SqlServer.Management.Smo.Restore.ReadMediaHeader(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To view backup header information

- [RESTORE HEADERONLY \(Transact-SQL\)](#)

- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)

- [How to: View the Properties and Contents of a Backup Device \(SQL Server Management Studio\)](#)

-

M:Microsoft.SqlServer.Management.Smo.Restore.ReadBackupHeader(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To delete old rows from backup and restore history tables

- [sp_delete_backuphistory](#)

To delete all rows for a specific database from backup and restore history tables

- [sp_delete_database_backuphistory](#)

To view media header information

- [RESTORE LABELONLY \(Transact-SQL\)](#)
- [How to: View the Properties and Contents of a Backup Device \(SQL Server Management Studio\)](#)
- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Restore.ReadMediaHeader(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To view backup header information

- [RESTORE HEADERONLY \(Transact-SQL\)](#)
- [How to: View the Contents of a Backup Tape or File \(SQL Server Management Studio\)](#)
- [How to: View the Properties and Contents of a Backup Device \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Restore.ReadBackupHeader(Microsoft.SqlServer.Management.Smo.Server) (SMO)

To view the files in a backup set

- [How to: View the Data and Log Files In a Backup Set \(SQL Server Management Studio\)](#)
- [RESTORE HEADERONLY \(Transact-SQL\)](#)

To verify a backup

- [RESTORE VERIFYONLY \(Transact-SQL\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Restore.SqlVerify(Microsoft.SqlServer.Management.Smo.Server) (SMO)



See Also

[BACKUP \(Transact-SQL\)](#)

[Media Sets, Media Families, and Backup Sets \(SQL Server\)](#)

[Backup Devices \(SQL Server\)](#)

[Mirrored Backup Media Sets](#)

[Possible Media Errors During Backup and Restore \(SQL Server\)](#)

View the Properties and Contents of a Logical Backup Device

This topic describes how to view the properties and contents of a logical backup device in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To view the properties and contents of a logical backup device, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

For information about security, see [RESTORE LABELONLY \(Transact-SQL\)](#).

Permissions

In SQL Server 2008 and later versions, obtaining information about a backup set or backup device requires CREATE DATABASE permission. For more information, see [GRANT Database Permissions \(Transact-SQL\)](#).



Using SQL Server Management Studio

To view the properties and contents of a logical backup device

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Server Objects**, and expand **Backup Devices**.
3. Click the device and right-click **Properties**, which opens the **Backup Device** dialog box.
4. The **General** page displays the device name and destination, which is either a tape device or file path.
5. In the **Select a Page** pane, click **Media Contents**.
6. The right-hand pane displays in the following properties panels:
 - **Media**
Media sequence information (the media sequence number, the family sequence number, and the mirror identifier, if any) and the media creation

date and time.

- **Media set**

Media set information: the media set name and description, if any, and the number of families in the media set.

7. The **Backup sets** grid displays information about the contents of the media set.



Note

For more information, see [Media Contents Page](#).



Using Transact-SQL

▶ To view the properties and contents of a logical backup device

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Use the [RESTORE LABELONLY](#) statement. This example returns information about the AdvWrks2008R2Backup logical backup device.

```
USE AdventureWorks2012 ;  
RESTORE LABELONLY  
FROM AdvWrks2008R2Backup ;  
GO
```



See Also

[backupfilegroup \(Transact-SQL\)](#)

[backupfile \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[backupmediaset \(Transact-SQL\)](#)

[backupmediafamily \(Transact-SQL\)](#)

[sp_addumpdevice \(Transact-SQL\)](#)

[Backup Devices \(SQL Server\)](#)

Possible Media Errors During Backup and Restore

SQL Server 2012 gives you the option of recovering a database despite detected errors. An important new error-detection mechanism is the optional creation of a backup checksum that can be created by a backup operation and validated by a restore operation. You can control whether an operation checks for errors and whether the operation stops or continues on encountering an error. If a backup contains a backup checksum, RESTORE and RESTORE VERIFYONLY statements can check for errors.

Note

Mirrored backups provide up to four copies (mirrors) of a media set, providing alternative copies for recovering from errors caused by damaged media. For more information, see [Mirrored Backup Media Sets](#).

In this Topic:

- Backup Checksums
- Response to Page Checksum Errors During a Backup or Restore Operation
- Related Tasks

Backup Checksums

SQL Server supports three types of checksums: a checksum on pages, a checksum in log blocks, and a backup checksum. When generating a backup checksum, BACKUP verifies that the data read from the database is consistent with any checksum or torn-page indication that is present in the database.

The BACKUP statement optionally computes a backup checksum on the backup stream; if page-checksum or torn-page information is present on a given page, when backing up the page, BACKUP also verifies the checksum and torn-page status and the page ID, of the page. When creating a backup checksum, a backup operation does not add any checksums to pages. Pages are backed up as they exist in the database, and the pages are unmodified by backup.

Due to the overhead verifying and generating backup checksums, using backup checksums poses a potential performance impact. Both the workload and the backup throughput may be affected. Therefore, using backup checksums is optional. When deciding to generate checksums during a backup, carefully monitor the CPU overhead incurred as well as the impact on any concurrent workload on the system.

BACKUP never modifies the source page on disk nor the contents of a page.

When backup checksums are enabled, a backup operation performs the following steps:

1. Before writing a page to the backup media, the backup operation verifies the page-level information (page checksum or torn page detection), if either exists. If neither

exists, backup cannot verify the page. Unverified the pages are included as is, and their contents are added to the overall backup checksum.

If the backup operation encounters a page error during verification, the backup fails.

 **Note**

For more information about page checksums and torn page detection, see the PAGE_VERIFY option of the ALTER DATABASE statement. For more information, see [ALTER DATABASE SET Options \(Transact-SQL\)](#).

2. Regardless of whether page checksums are present, BACKUP generates a separate backup checksum for the backup streams. Restore operations can optionally use the backup checksum to validate that the backup is not corrupted. The backup checksum is stored on the backup media, not on the database pages. The backup checksum can optionally be used at restore time.
3. The backup set is flagged as containing backup checksums (in the **has_backup_checksums** column of **msdb..backupset**). For more information, see [backupset \(Transact-SQL\)](#).

During a restore operation, if backup checksums are present on the backup media, by default, both the RESTORE and RESTORE VERIFYONLY statements verify the backup checksums and page checksums. If there is no backup checksum, either restore operation proceeds without any verification; this is because without a backup checksum, restore cannot reliably verify page checksums.



Response to Page Checksum Errors During a Backup or Restore Operation

By default, after encountering a page checksum error, a BACKUP or RESTORE operation fails and a RESTORE VERIFYONLY operation continues. However, you can control whether a given operation fails on encountering an error or continues as best it can.

If a BACKUP operation continues after encountering errors, the operation performs the following steps:

1. Flags the backup set on the backup media as containing errors and tracks the page in the **suspect_pages** table in the **msdb** database. For more information, see [suspect_pages \(Transact-SQL\)](#).
2. Logs the error in the SQL Server error log.
3. Marks the backup set as containing this type of error (in the **is_damaged** column of **msdb.backupset**). For more information, see [backupset \(Transact-SQL\)](#).
4. Issues a message that the backup was successfully generated, but contains page errors.



Related Tasks

To enable or disable backup checksums

- [How to: Enable or Disable Backup Checksums \(Transact-SQL\)](#)

To control the response to an error during a backup operation

- [How to: Specify Whether BACKUP Continues or Stops upon Encountering an Error \(Transact-SQL\)](#)



See Also

[ALTER DATABASE \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[Using Mirrored Backup Media Sets](#)

[RESTORE \(Transact-SQL\)](#)

[RESTORE VERIFYONLY \(Transact-SQL\)](#)

Enable or Disable Backup Checksums During Backup or Restore

This topic describes how to enable or disable backup checksums when you are backing up or restoring a database in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To enable or disable backup checksums, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

Permissions

BACKUP

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin**

fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.

RESTORE

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To enable or disable checksums during a backup operation

1. Follow the steps to [create a database backup](#).
2. On the **Options** page, in the **Reliability** section, click **Perform checksum before writing to media**.



Using Transact-SQL

▶ To enable or disable backup checksum for a backup operation

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. To enable backup checksums in a [BACKUP](#) statement, specify the WITH CHECKSUM option. To disable backup checksums, specify the WITH NO_CHECKSUM option. This is the default behavior, except for a compressed backup. The following example specifies that checksums be performed.

```
BACKUP DATABASE AdventureWorks2012
```

```
TO DISK = 'Z:\SQLServerBackups\AdvWorksData.bak'  
    WITH CHECKSUM;  
GO
```

To enable or disable backup checksum for a restore operation

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. To enable backup checksums in a [RESTORE](#) statement, specify the WITH CHECKSUM option. This is the default behavior for a compressed backup. To disable backup checksums, specify the WITH NO_CHECKSUM option. This is the default behavior, except for a compressed backup. The following example specifies that backup checksums be performed.

```
RESTORE DATABASE AdventureWorks2012  
    FROM DISK = 'Z:\SQLServerBackups\AdvWorksData.bak'  
    WITH CHECKSUM;  
GO
```

Warning

If you explicitly request CHECKSUM for a restore operation and if the backup contains backup checksums, backup checksums and page checksums are both verified, as in the default case. However, if the backup set lacks backup checksums, the restore operation fails with a message indicating that checksums are not present.



See Also

[RESTORE FILELISTONLY \(Transact-SQL\)](#)

[RESTORE HEADERONLY \(Transact-SQL\)](#)

[RESTORE LABELONLY \(Transact-SQL\)](#)

[RESTORE VERIFYONLY \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[RESTORE Arguments \(Transact-SQL\)](#)

[Possible Media Errors During Backup and Restore \(SQL Server\)](#)

[Specify Whether BACKUP Continues or Stops upon Encountering an Error \(Transact-SQL\)](#)

Specify Whether a Backup or Restore Operation Continues or Stops After Encountering an Error

This topic describes how to specify whether a backup or restore operation continues or stops after encountering an error in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Security
- **To specify whether a backup or restore operation continues after encountering an error, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Security

Permissions

BACKUP

BACKUP DATABASE and BACKUP LOG permissions default to members of the **sysadmin** fixed server role and the **db_owner** and **db_backupoperator** fixed database roles.

Ownership and permission problems on the backup device's physical file can interfere with a backup operation. SQL Server must be able to read and write to the device; the account under which the SQL Server service runs must have write permissions.

However, [sp_addumpdevice](#), which adds an entry for a backup device in the system tables, does not check file access permissions. Such problems on the backup device's physical file may not appear until the physical resource is accessed when the backup or restore is attempted.

RESTORE

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case

when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To specify whether backup continues or stops after an error is encountered

1. Follow the steps to [create a database backup](#).
2. On the **Options** page, in the **Reliability** section, click **Perform checksum before writing to media** and **Continue on error**.



Using Transact-SQL

▶ To specify whether a backup operation continues or stops after encountering an error

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. In the [BACKUP](#) statement, specify the CONTINUE_AFTER_ERROR option to continue or the STOP_ON_ERROR option to stop. The default behavior is to stop after encountering an error. This example instructs the backup operation to continue despite encountering an error.

```
BACKUP DATABASE AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdvWorksData.bak'
WITH CHECKSUM, CONTINUE_AFTER_ERROR;
GO
```

▶ To specify whether a restore operation continues or stops after encountering an error

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. In the [RESTORE](#) statement, specify the CONTINUE_AFTER_ERROR option to continue or the STOP_ON_ERROR option to stop. The default behavior is to stop after encountering an error. This example instructs the restore operation to continue despite encountering an error.

```
RESTORE DATABASE AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdvWorksData.bak'
WITH CHECKSUM, CONTINUE_AFTER_ERROR;
```

GO



See Also

[RESTORE FILELISTONLY \(Transact-SQL\)](#)

[RESTORE HEADERONLY \(Transact-SQL\)](#)

[RESTORE LABELONLY \(Transact-SQL\)](#)

[RESTORE VERIFYONLY \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[backupset \(Transact-SQL\)](#)

[RESTORE Arguments \(Transact-SQL\)](#)

[Detecting and Coping with Media Errors](#)

[How to: Enable or Disable Backup Checksums \(Transact-SQL\)](#)

Complete Database Restores (Simple Recovery Model)

In a complete database restore, the goal is to restore the whole database. The whole database is offline for the duration of the restore. Before any part of the database can come online, all data is recovered to a consistent point in which all parts of the database are at the same point in time and no uncommitted transactions exist.

Under the simple recovery model, the database cannot be restored to a specific point in time within a specific backup.

noteDXDOC112778PADS **Security Note**

We recommend that you do not attach or restore databases from unknown or untrusted sources. These databases could contain malicious code that might execute unintended Transact-SQL code or cause errors by modifying the schema or the physical database structure. Before you use a database from an unknown or untrusted source, run [DBCC CHECKDB](#) on the database on a nonproduction server and also examine the code, such as stored procedures or other user-defined code, in the database.

In this Topic:

- Overview of Database Restore Under the Simple Recovery Model
- Related Tasks

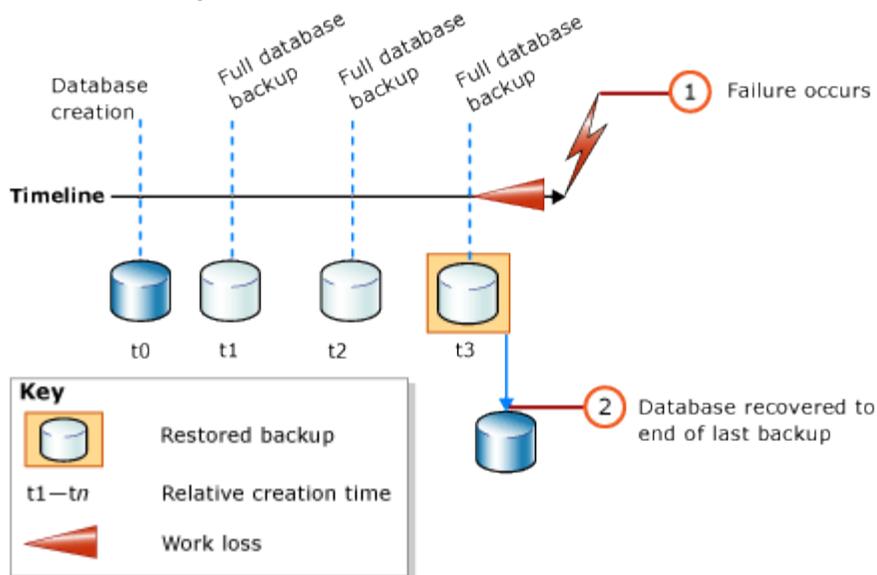


Note

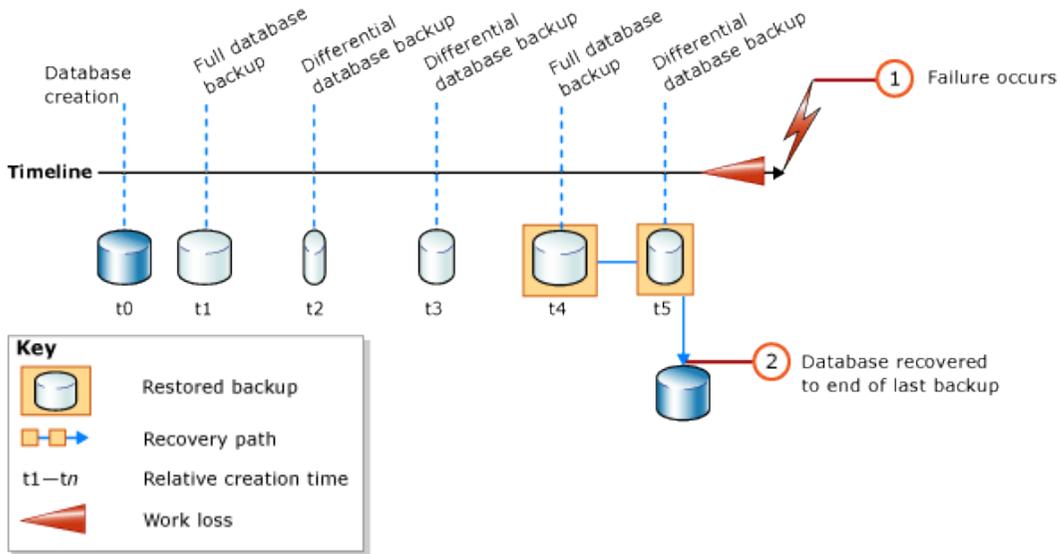
For information about support for backups from earlier versions of SQL Server, see the "Compatibility Support" section of [RESTORE \(Transact-SQL\)](#).

Overview of Database Restore Under the Simple Recovery Model

A full database restore under the simple recovery model involves one or two [RESTORE](#) statements, depending on whether you want to restore a differential database backup. If you are using only a full database backup, just restore the most recent backup, as shown in the following illustration.



If you are also using a differential database backup, restore the most recent full database backup without recovering the database, and then restore the most recent differential database backup and recover the database. The following illustration shows this process.



Note

If you plan to restore a database backup onto a different server instance, see [Copying Databases with Backup and Restore](#).

Basic Transact-SQL RESTORE Syntax

The basic Transact-SQL [RESTORE](#) syntax for restoring a full database backup is:
`RESTORE DATABASE database_name FROM backup_device [WITH NORECOVERY]`



Note

Use WITH NORECOVERY if you plan to also restore a differential database backup.

The basic [RESTORE](#) syntax for restoring a database backup is:

`RESTORE DATABASE database_name FROM backup_device WITH RECOVERY`

Example (Transact-SQL)

The following example first shows how to use the [BACKUP](#) statement to create a full database backup and a differential database backup of the database. The example then restores these backups in sequence. The database is restored to its state as of the time that the differential database backup finished.

The example shows the critical options in a restore sequence for the complete database restore scenario. A *restore sequence* consists of one or more restore operations that move data through one or more of the phases of restore. Syntax and details that are not relevant to this purpose are omitted. When you recover a database, we recommend explicitly specifying the RECOVERY option for clarity, even though it is the default.

 **Note**

The example starts with an [ALTER DATABASE](#) statement that sets the recovery model to SIMPLE.

```
USE master;

--Make sure the database is using the simple recovery model.
ALTER DATABASE AdventureWorks2012 SET RECOVERY SIMPLE;
GO

-- Back up the full AdventureWorks2012 database.
BACKUP DATABASE AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak'
    WITH FORMAT;
GO

--Create a differential database backup.
BACKUP DATABASE AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak'
    WITH DIFFERENTIAL;
GO

--Restore the full database backup (from backup set 1).
RESTORE DATABASE AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak'
    WITH FILE=1, NORECOVERY;

--Restore the differential backup (from backup set 2).
RESTORE DATABASE AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdventureWorks2012.bak'
    WITH FILE=2, RECOVERY;
GO
```

Related Tasks

To restore a full database backup

- [Restore a Full Database Backup \(Transact-SQL\)](#)
- [Restore a Database Backup \(SQL Server Management Studio\)](#)
- [Create a New Database From An Existing Database Backup \(SQL Server Management Studio\)](#)

To restore a differential database backup

- [Restore a Full Differential Backup \(SQL Server Management Studio\)](#)

To restore a backup by using SQL Server Management Objects (SMO)

-

```
M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.M  
anagement.Smo.Server)
```

See Also

[RESTORE \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[sp_addumpdevice \(Transact-SQL\)](#)

[Full Database Backups](#)

[Differential Backups \(SQL Server\)](#)

[Backup Overview \(SQL Server\)](#)

[Restore and Recovery Overview \(SQL Server\)](#)

Restore a Database Backup Under the Simple Recovery Model (Transact-SQL)

This topic explains how to restore a full database backup.

Important

The system administrator restoring the full database backup must be the only person currently using the database to be restored.

Prerequisites and Recommendations

- To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).
- For security purposes, we recommend that you do not attach or restore databases from unknown or untrusted sources. Such databases could contain malicious code that might execute unintended Transact-SQL code or cause errors by modifying the schema or the physical database structure. Before you use a database from an unknown or untrusted source, run [DBCC CHECKDB](#) on the database on a nonproduction server and also examine the code, such as stored procedures or other user-defined code, in the database.

Database Compatibility Level After Upgrade

The compatibility levels of the **tempdb**, **model**, **msdb** and **Resource** databases are set to 100 after upgrade. The **master** system database retains the compatibility level it had before upgrade, unless that level was less than 90. If the compatibility level of **master** was less than 90 before upgrade, it is set to 90 after upgrade.

If the compatibility level of a user database was 90 or 100 before upgrade, it remains the same after upgrade. If the compatibility level was 80 or less before upgrade, in the upgraded database, the compatibility level is set to 90, which is the lowest supported compatibility level in SQL Server 2012.

Note

New user databases will inherit the compatibility level of the **model** database.

Procedures

To restore a full database backup

1. Execute the RESTORE DATABASE statement to restore the full database backup, specifying:
 - The name of the database to restore.
 - The backup device from where the full database backup is restored.
 - The NORECOVERY clause if you have a transaction log or differential database backup to apply after restoring the full database backup.

Important

To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

2. Optionally, specify:
 - The FILE clause to identify the backup set on the backup device to restore.

Note

- If you restore a SQL Server 2005 or SQL Server 2008 database to SQL Server 2012, the database is automatically upgraded. Typically, the database becomes available immediately. However, if a SQL Server 2005 database has full-text indexes, the upgrade process either imports, resets, or rebuilds them, depending on the setting of the **upgrade_option** server property. If the upgrade option is set to import (**upgrade_option** = 2) or rebuild (**upgrade_option** = 0), the full-text indexes will be unavailable during the upgrade. Depending the amount of data being indexed, importing can take

several hours, and rebuilding can take up to ten times longer. Note also that when the upgrade option is set to import, the associated full-text indexes are rebuilt if a full-text catalog is not available. To change the setting of the **upgrade_option** server property, use [sp_fulltext_service](#).

Example

Description

This example restores the full database backup from tape.

Code

```
USE master
GO
RESTORE DATABASE AdventureWorks2012
    FROM TAPE = '\\.\Tape0'
GO
```

See Also

[Performing a Complete Database Restore \(Full Recovery Model\)](#)

[Performing a Complete Database Restore \(Simple Recovery Model\)](#)

[Full Database Backups](#)

[RESTORE](#)

[Viewing Information About Backups](#)

[Rebuilding System Databases](#)

Restore a Database Backup (SQL Server Management Studio)

This topic explains how to restore a full database backup.

Important

Under the full or bulk-logged recovery model, before you can restore a database in SQL Server Management Studio, you must back up the active transaction log (known as the tail of the log). For more information, see [SQL Server Management Studio Tutorial](#). To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

Note that if you restore a SQL Server 2005 or SQL Server 2008 database to SQL Server 2012, the database is automatically upgraded. Typically, the database becomes available immediately. However, if a SQL Server 2005 database has full-text indexes, the upgrade process either imports, resets, or rebuilds them, depending on the setting of the **Full-Text Upgrade Option** server property. If the upgrade option is set to **Import** or **Rebuild**, the full-text indexes will be unavailable during the upgrade. Depending upon the amount of data being indexed, importing can take several hours, and rebuilding can take up to ten times longer. Note also that when the upgrade option is set to **Import**, if a full-text catalog is not available, the associated full-text indexes are rebuilt. For information about viewing or changing the setting of the **Full-Text Upgrade Option** property, see [Manage and Monitor Full-Text Search for a Server Instance](#).

Procedures

▶ To restore a full database backup

1. After you connect to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases**, and then select a system database.
3. Right-click the database, point to **Tasks**, point to **Restore**, and then click **Database**, which opens the **Restore Database** dialog box.
4. On the **General** page, use the **Source** section to specify the source and location of the backup sets to restore. Select one of the following options:

- **Database**

Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.



Note

If the backup is taken from a different server, the destination server will not have the backup history information for the specified database. In this case, select **Device** to manually specify the file or device to restore.

- **Device**

Click the browse (...) button to open the **Select backup devices** dialog box.

In the **Backup media type** box, select one of the listed device types. To select one or more devices for the **Backup media** box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.

In the **Source: Device: Database** list box, select the name of the database which should be restored.

 **Note**

This list is only available when **Device** is selected. Only databases that have backups on the selected device will be available.

Backup media

Select the medium for the restore operation: **File**, **Tape**, or **Backup Device**. The **Tape** option appears only if a tape drive is mounted on the computer, and the **Backup Device** option appears, only if at least one backup device exists.

Backup location

View, add, or remove media for the restore operation. The list can contain up to 64 files, tapes, or backup devices.

Add

Adds the location of a backup device to the **Backup location** list. Depending on the type of media you select in the **Backup media** field, clicking **Add** opens one of the following dialog boxes.

Media type	Dialog box	Description
File	Locate Backup File	In this dialog box, you can select a local file from the tree or specify a remote file using its fully qualified universal naming convention (UNC) name. For more information, see Backup Devices .
Device	Select Backup Device	In this dialog box, you can select from a list of the logical backup devices defined on the server instance.
Tape	Select Backup Tape	In this dialog box, you can select from a list of the tape drives that are physically connected to the computer running the instance of SQL Server.

If the list is full, the **Add** button is unavailable.

Remove

Removes one or more selected files, tapes, or logical backup devices.

Contents

Displays the media contents of a selected file, tape, or logical backup device.

5. In the **Destination** section, the **Database** box is automatically populated with the name of the database to be restored. To change the name of the database, enter the new name in the **Database** box.
6. In the **Restore to** box, leave the default as **To the last backup taken** or click on **Timeline** to access the **Backup Timeline** dialog box to manually select a point in time to stop the recovery action. For more information on designating a specific point in time, see [Backup Timeline](#).
7. In the **Backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Backups that depend on the restoration of an earlier backup are automatically deselected when the earlier backup is deselected. For information about the columns in the **Backup sets to restore** grid, see [Restore Database \(General Page\)](#).
8. Optionally, click **Files** in the **Select a page** pane to access the **Files** dialog box. From here, you can restore the database to a new location by specifying a new restore destination for each file in the **Restore the database files as** grid. For more information about this grid, see [Restore Database \(Database Files Page\)](#).
9. To view or select the advanced options, on the **Options** page, in the **Restore options** panel, you can select any of the following options, if appropriate for your situation:
 - a. **WITH** options (not required):
 - **Overwrite the existing database (WITH REPLACE)**
 - **Preserve the replication settings (WITH KEEP_REPLICATION)**
 - **Restrict access to the restored database (WITH RESTRICTED_USER)**
 - b. Select an option for the **Recovery state** box. This box determines the state of the database after the restore operation.
 - **RESTORE WITH RECOVERY** is the default behavior which leaves the database ready for use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored. Select this option if you are restoring all of the necessary backups now.
 - **RESTORE WITH NORECOVERY** which leaves the database non-operational, and does not roll back the uncommitted transactions. Additional transaction logs can be restored. The database cannot be used until it is recovered.

- **RESTORE WITH STANDBY** which leaves the database in read-only mode. It undoes uncommitted transactions, but saves the undo actions in a standby file so that recovery effects can be reverted.
- c. **Take tail-log backup before restore** will be selected if it is necessary for the point in time that you have selected. You do not need to modify this setting, but you can choose to backup the tail of the log even if it is not required.
- d. Restore operations may fail if there are active connections to the database. Check the **Close existing connections option** to ensure that all active connections between Management Studio and the database are closed. This check box sets the database to single user mode before performing the restore operations, and sets the database to multi-user mode when complete.
- e. Select **Prompt before restoring each backup** if you wish to be prompted between each restore operation. This is not usually necessary unless the database is large and you wish to monitor the status of the restore operation.

For more information about these restore options, see [Restore Database \(Options Page\)](#).

10. Click .

See Also

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Back Up a Database \(SQL Server Management Studio\)](#)

[How to: Create a New Database From An Existing Full Backup \(SQL Server Management Studio\)](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[RESTORE \(Transact-SQL\)](#)

[Restore Database \(Options Page\)](#)

[Restore Database \(General Page\)](#)

Backup Timeline

Use the **Backup Timeline** dialog box to locate and specify backups to restore a database to a point-in-time. The **Backup Timeline** dialog box is accessed by clicking **Timeline** on the **Restore Database (General Page)** pane. This dialog box allows you to view a timeline of the restore operations performed on the database.

Restore to

Last backup taken is selected by default. SQL Server Management Studio will select the appropriate backups to restore the database, and will restore the database to the point of the last backup. Click **A specific date and time** to manually set the date and time (selecting a specific point in time).

Specific date and time permits you stop the restore at a specific date and time that you select. The timeline shows a representation of the backup operations performed in the 24 hours around the select date and time.

Date

Enter or select a date from the drop-down list.

Time

Enter or select a date to designate the specific point-in-time for the restore to stop.

Timeline Interval

Displays the options for the interval types viewable on the timeline.

Timeline and Legend

Use the scroll bar beneath the timeline to move the cursor forward and backward along the timeline. Click on a backup to move the scroll bar to the end of that backup. Hover the mouse over a marker to display a ScreenTip providing information about the selected backup set. Backup information is shown on the timeline by the following markers.

Larger triangle

Represents the full backups performed on the database, denoting the specific point in time each full backup was performed.

Smaller triangle

Represents the differential backups performed on the database, denoting the specific point in time that each differential backup was performed.

Green shaded areas

Represents the transaction log backup coverage.

Red line

Can only be positioned along the timeline where the restore is possible. Moving the red line along the timeline adjusts the date and time displayed in the **Date** and **Time** boxes.

See Also

[Restore Database \(General Page\)](#)

Restore Database (General Page)

Use the **General** page to specify information about the target and source databases for a database-restore operation.

To use SQL Server Management Studio to restore a database backup

- [Working with Transaction Log Backups](#)
- [How to: Restore a Differential Database Backup \(SQL Server Management Studio\)](#)

Note

When you specify a restore task by using SQL Server Management Studio, you can generate the corresponding Transact-SQL [RESTORE](#) script by clicking **Script** and then selecting a destination for the script.

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database.

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.

Options

Source

The options of the **Restore from** panel identify the location of the backup sets for the database and which backup sets you want to restore.

Term	Definition
Database	Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the msdb backup history.
Device	Select the logical or physical backup devices (tapes or files) that contain the backup or backups you want to restore. This is required if the database backup was taken on a different instance of SQL Server. To select one or more logical or physical backup devices, click the browse button which opens the Select backup devices dialog box. There, you can select up to 64 devices that belong to a single media set. Tape devices must be physically connected to the computer that is running the instance of SQL Server. A backup file can be on a local or remote disk device. For more information, see Backup Devices .

Term	Definition
	When you exit the Select backup devices dialog box, the selected device will appear as read-only values in the Device list.
Database	<p>Select the database name from which the backups should be restored from the dropdown list.</p> <p> Note This list is only available when Device is selected. Only databases that have backups on the selected devices will be available.</p>

Destination

The options of the **Restore to** panel identify the database and restore point.

Term	Definition
Database	<p>Enter the database to restore in the list. You can enter a new database or choose an existing database from the drop-down list. The list includes all databases on the server, excluding the system databases master and tempdb.</p> <p> Note To restore a password-protected backup, you must use the RESTORE statement.</p>
Restore to	<p>The Restore to box will be set "To the last backup taken" by default. You can also click Timeline to show the Backup Timeline dialog box, which displays the database backup history in the form of a timeline. Click Timeline to designate a specific datetime to which you want to restore the database. The database will then be restored to the state it was in at this specified point in time. See Backup</p>

Term	Definition
	Timeline .

Restore Plan

Term	Definition								
Backup sets to restore	<p>Displays the backup sets available for the specified location. Each backup set, the result of a single backup operation, is distributed across all of the devices in the media set. By default, a recovery plan is suggested to achieve the goal of the restore operation that is based on the selection of the required backup sets. SQL Server Management Studio uses the backup history in msdb to identify which backups are required to restore a database, and creates a restore plan. For example, for a database restore, the restore plan selects the most recent full database backup followed by the most recent subsequent differential database backup, if any. Under the full recovery model, the restore plan then selects all subsequent log backups.</p> <p>To override the suggested recovery plan, you can change the selections in the grid. Any backups that depend on a deselected backup are deselected automatically.</p> <table border="1"> <thead> <tr> <th>Header</th> <th>Values</th> </tr> </thead> <tbody> <tr> <td>Restore</td> <td>The selected check boxes indicate the backup sets to be restored.</td> </tr> <tr> <td>Name</td> <td>The name of the backup set.</td> </tr> <tr> <td>Component</td> <td>The backed-up component:</td> </tr> </tbody> </table>	Header	Values	Restore	The selected check boxes indicate the backup sets to be restored.	Name	The name of the backup set.	Component	The backed-up component:
Header	Values								
Restore	The selected check boxes indicate the backup sets to be restored.								
Name	The name of the backup set.								
Component	The backed-up component:								

Term	Definition	
		Database, File, or <blank> (for transaction logs).
	Type	The type of backup performed: Full, Differential, or Transaction Log.
	Server	The name of the Database Engine instance that performed the backup operation.
	Database	The name of the database involved in the backup operation.
	Position	The position of the backup set in the volume.
	First LSN	The log sequence number of the first transaction in the backup set. Blank for file backups.
	Last LSN	The log sequence number of the last transaction in the backup set. Blank for file backups.
	Checkpoint LSN	The log sequence number (LSN) of the most recent checkpoint at the time the backup was created.
	Full LSN	The log sequence number of the most

Term	Definition	
		recent full database backup.
	Start Date	The date and time when the backup operation began, presented in the regional setting of the client.
	Finish Date	The date and time when the backup operation finished, presented in the regional setting of the client.
	Size	The size of the backup set in bytes.
	User Name	The name of the user who performed the backup operation.
	Expiration	The date and time the backup set expires.
	<p>The checkboxes are only enabled when the Manual Selection box is checked. This allows you to select which backup-sets are to be restored.</p> <p>When the Manual Selection box is checked, the accuracy of the Restore Plan is checked each time it is modified. If the sequence of backups is incorrect, an error message will appear.</p>	
Verify Backup Media	Calls a RESTORE VERIFY_ONLY statement on the selected backup-sets.	

Term	Definition
	 Note This is a long-running operation, and its progress can be tracked and cancelled by using the Progress Monitor on the Dialog Framework. This button allows you to check the integrity of the selected backup files prior to restoring them. When checking the integrity of backup sets, the progress status at the bottom left of the dialog box will read "Verifying" rather than "Executing."

Compatibility Support

In SQL Server 2012, you can restore a user database from a database backup that was created by using SQL Server 2005 or a later version. However, backups of **master**, **model** and **msdb** that were created by using SQL Server 2005 or SQL Server 2008 cannot be restored by SQL Server 2012. Also, backups created in SQL Server 2012 cannot be restored by any earlier version of SQL Server.

Note

No SQL Server backup be restored to an earlier version of SQL Server than the version on which the backup was created.

SQL Server 2012 uses a different default path than earlier versions. Therefore, to restore a database that was created in the default location for SQL Server 2005 or SQL Server 2008 backups, you must use the MOVE option.

After you restore a SQL Server 2005 or SQL Server 2008 database to SQL Server 2012, the database is automatically upgraded. Typically, the database becomes available immediately. However, if a SQL Server 2005 database has full-text indexes, the upgrade process either imports, resets, or rebuilds them, depending on the setting of the **Full-Text Upgrade Option** server property. If the upgrade option is set to **Import** or **Rebuild**, the full-text indexes will be unavailable during the upgrade. Depending upon the amount of data being indexed, importing can take several hours, and rebuilding can take up to ten times longer. Note also that when the upgrade option is set to **Import**, if a full-text catalog is not available, the associated full-text indexes are rebuilt.

See Also

[Backup Devices](#)

[How to: Restore a Backup from a Device \(SQL Server Management Studio\)](#)

[How to: Restore a Marked Transaction \(SQL Server Management Studio\)](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[How to: View the Content of a Backup Tape or File \(SQL Server Management Studio\)](#)

[How to: View the Properties and Content of a Logical Backup Device \(SQL Server Management Studio\)](#)

[Media Sets, Media Families, and Backup Sets](#)

[RESTORE Arguments \(Transact-SQL\)](#)

[Apply Transaction Log Backups](#)

Restore Database (Options Page)

Use the **Options** page of the **Restore Database** dialog box to modify the behavior and outcome of the restore operation.

To use SQL Server Management Studio to restore a database backup

- [RESTORE \(Transact-SQL\)](#)
- [How to: Restore a Differential Database Backup \(SQL Server Management Studio\)](#)



Note

When you specify a restore task by using SQL Server Management Studio, you can generate a corresponding Transact-SQL script containing the RESTORE statements for this restore operation. To generate the script, click **Script** and then select a destination for the script. For information about the RESTORE syntax, see [RESTORE \(Transact-SQL\)](#).

Options

Restore options

To modify aspects of the behavior of the restore operation, use the options of the **Restore options** panel.

Overwrite the existing database [WITH REPLACE]

The restore operation will overwrite the files of any database that is currently using the database name that you are specifying in the **Restore to** field on the [General](#) page of the **Restore Database** dialog box. The files of the existing database will be overwritten even if you are restoring backups from a different database to the existing database name. Selecting this option is equivalent to using the REPLACE option in a [RESTORE](#) statement (Transact-SQL).



Caution

Use this option only after careful consideration. For more information, see [RESTORE Arguments \(Transact-SQL\)](#).

Preserve the replication settings [WITH KEEP_REPLICATION]

Preserves the replication settings when restoring a published database to a server other

than the server where the database was created. This option is relevant only if the database was replicated when the backup was created.

This option is available only with the **Leave the database ready for use by rolling back the uncommitted transactions** option (described later in this table), which is equivalent to restoring a backup with the RECOVERY option.

Selecting this option is equivalent to using the KEEP_REPLICATION option in a [RESTORE](#) statement.

For more information, see [Backing Up and Restoring Replicated Databases](#).

Restrict access to the restored database [WITH RESTRICTED_USER]

Makes the restored database available only to the members of **db_owner**, **dbcreator**, or **sysadmin**.

Selecting this option is synonymous to using the RESTRICTED_USER option in a RESTORE statement.

Recovery state

To determine the state of the database after the store operation, you must select one of the options of the **Recovery state** panel.

RESTORE WITH RECOVERY

Recovers the database after restoring the final backup checked in the **Backup sets to restore** grid on the [General page](#). This is the default option and is equivalent to specifying WITH RECOVERY in a [RESTORE](#) statement (Transact-SQL).



Note

Under the full recovery model or bulk-logged recovery model, choose this option only if you are restoring all the log files now.

RESTORE WITH NORECOVERY

Leaves the database in the restoring state. This allows you to restore additional backups in the current recovery path. To recover the database, you will have to perform a restore operation by using the RESTORE WITH RECOVERY option (see the preceding option).

This option is equivalent to specifying WITH NORECOVERY in a RESTORE statement.

If you select this option, the **Preserve replication settings** option is unavailable.

RESTORE WITH STANDBY

Leaves the database in a standby state, in which the database is available for limited read-only access. This option is equivalent to specifying WITH STANDBY in a RESTORE statement.

Choosing this option requires that you specify a standby file in the **Standby file** text box. The standby file allows the recovery effects to be undone.

Standby file

Specifies a standby file. You can browse for the standby file or enter its pathname

directly in the text box.

Tail-Log backup

Allows you to designate that a tail-log backup be performed along with the database restore.

Take tail-Log backup before restoring

Check this box to designate that a tail-log backup should be performed.



Note

- If the point-in-time you have selected in the [Backup Timeline](#) dialog box requires a tail-log backup, this box will be selected and you will not be able to edit it.

Backup file

Specifies a backup file for the tail of the log. You can browse for the backup file or enter its name directly in the text box.

Server connections

Allows you to close existing database connections.

Close existing connections

Restore operations may fail if there are active connections to the database. Check the **Close existing connections option** to ensure that all active connections between Management Studio and the database are closed. This check box sets the database to single user mode before performing the restore operations, and sets the database to multi-user mode when complete.

Prompt

Prompt before restoring each backup

Specifies that after each backup is restored, the **Continue with Restore** dialog box will be displayed to inquire whether you want to continue the restore sequence. This dialog box displays the name of the next media set (if known) and the name and description of the next backup set.

This option allows you to pause a restore sequence after restoring any of the backups. This option is particularly useful when you must swap tapes for different media sets; for example, when your server has only one tape device. When you are ready to proceed, click **OK**.

You can interrupt a restore sequence by clicking **No**. This leaves the database in the restoring state. At your convenience, you can later continue the restore sequence by resuming with the next backup described in the **Continue with Restore** dialog box. The procedure restoring the next backup depends on whether it contains data or transaction log, as follows:

- If the next backup is a full or differential backup, use the **Restore Database** task again.

- If the next backup is a file backup, use the **Restore Files and Filegroups** task. For more information, see [How to: Restore Files and Filegroups \(SQL Server Management Studio\)](#).
- If the next backup is a log backup, use the **Restore Transaction Log** task. For information about resuming a restore sequence by restoring a transaction log, see [How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#).

See Also

[RESTORE \(Transact-SQL\)](#)

[Restore a Backup from a Device \(SQL Server\)](#)

[Restore a Transaction Log Backup \(SQL Server\)](#)

[Media Sets, Media Families, and Backup Sets](#)

[Apply Transaction Log Backups](#)

[Restore Database \(General Page\)](#)

Restore Database (Files Page)

Use the **Files** page of the **Restore Database** dialog box to manage the specific files you have chosen to restore within the database.

Options

Restore database files as

Use to assign and manage the new file path to the restored files.

Relocate all files to folder

Relocates restored files.

Option	Description
Data file folder	Enter or search for the data file folder name that the restored data file or files should be relocated to.
Log file folder	Enter or search for the log file or files folder that the restored log file should be relocated to.

Logical File Name

Displays one row for each database file to be restored.

File Type

Displays the file type.

Original File Name

Displays the original file path for the restored file.

Restore As

Lists the file names that the restored files are to be saved as. Enter or search for the appropriate file name.

See Also

[Restore Database \(General Page\)](#)

[Restore Database \(Options Page\)](#)

[RESTORE Arguments \(Transact-SQL\)](#)

[How to: Restore a Differential Database Backup \(SQL Server Management Studio\)](#)

[RESTORE \(Transact-SQL\)](#)

Continue with Restore

Use the **Continue with Restore** dialog box to indicate whether you want to restore the next backup set. To delay the restore operation, for example, to swap tapes, wait until you are ready to proceed before you click **OK**.

Clicking **No** terminates the restore sequence, leaving the database in the restoring state. To continue with the restore later, use either the **Restore Database** or **Restore Transaction Log** task, as appropriate.

Options**Media set**

Displays the next media set name (if available).

Backup set

Displays the backup set name.

Backup set description

Displays the backup set description.

See Also

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[How to: View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#)

[How to: Restore a Database Backup \(SQL Server Management Studio\)](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

Select Backup Device

Use the **Select Backup Device** dialog box to select a logical backup device for the restore operation.

A logical backup device is a user-defined logical device that corresponds to a physical device, either a tape drive or a disk drive, that is provided by the operating system.

Note

If a backup uses multiple backup devices, they all must correspond to a single type of device.

To use SQL Server Management Studio to view the contents of a backup device

- [Backup Devices](#)
- [How to: View the Properties and Contents of a Logical Backup Device \(SQL Server Management Studio\)](#)

Options

Backup device

In the list box, select the name of a logical backup device from which you want to restore.

For information about how to view the contents of a backup device, see [How to: View the Properties and Content of a Backup Device \(SQL Server Management Studio\)](#).

Remarks

If you do not see a logical backup device that contains the backup you are seeking on the list, the backup might have been written directly to one or more files or tape drives. If this is the case, cancel the **Select Backup Device** dialog box; and in the **Specify Backup** dialog box, select **File** or **Tape** in the **Backup media** list box.

See Also

[Backup Devices](#)

Restore a Database to a New Location

This topic describes how to restore a SQL Server database to a new location, and optionally rename the database, in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. You can move a database to a new directory path or create a copy of a database on either the same server instance or a different server instance.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Prerequisites
 - Recommendations

Security

- **To restore a database to a new location, and optionally rename the database, using:**
 - SQL Server Management Studio
 - Transact-SQL
- Related Tasks

Before You Begin

Limitations and Restrictions

- The system administrator restoring a full database backup must be the only person currently using the database to be restored.

Prerequisites

- Under the full or bulk-logged recovery model, before you can restore a database, you must back up the active transaction log. For more information, see [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#).

Recommendations

- To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).
- For information about additional considerations for moving a database, see [Copying Databases with Backup and Restore](#).
- If you restore a SQL Server 2005 or SQL Server 2008 database to SQL Server 2012, the database is automatically upgraded. Typically, the database becomes available immediately. However, if a SQL Server 2005 database has full-text indexes, the upgrade process either imports, resets, or rebuilds them, depending on the setting of the **upgrade_option** server property. If the upgrade option is set to import (**upgrade_option** = 2) or rebuild (**upgrade_option** = 0), the full-text indexes will be unavailable during the upgrade. Depending the amount of data being indexed, importing can take several hours, and rebuilding can take up to ten times longer. Note also that when the upgrade option is set to import, the associated full-text indexes are rebuilt if a full-text catalog is not available. To change the setting of the **upgrade_option** server property, use [sp_fulltext_service](#).

Security

For security purposes, we recommend that you do not attach or restore databases from unknown or untrusted sources. Such databases could contain malicious code that might execute unintended Transact-SQL code or cause errors by modifying the schema or the physical database structure. Before you use a database from an unknown or untrusted source, run [DBCC CHECKDB](#) on the database on a nonproduction server and also examine the code, such as stored procedures or other user-defined code, in the database.

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database.

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore a database to a new location, and optionally rename the database

1. Connect to the appropriate instance of the SQL Server Database Engine, and then in Object Explorer, click the server name to expand the server tree.
2. Right-click **Databases**, and then click **Restore Database**. The **Restore Database** dialog box opens.
3. On the **General** page, use the **Source** section to specify the source and location of the backup sets to restore. Select one of the following options:

- **Database**

Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.



Note

If the backup is taken from a different server, the destination server will not have the backup history information for the specified database. In this case, select **Device** to manually specify the file or device to restore.

- a. **Device**

Click the browse (...) button to open the **Select backup devices** dialog box. In the **Backup media type** box, select one of the listed device types. To select one or more devices for the **Backup media** box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.

In the **Source: Device: Database** list box, select the name of the database which should be restored.

Note This list is only available when **Device** is selected. Only databases that have backups on the selected device will be available.

4. In the **Destination** section, the **Database** box is automatically populated with the name of the database to be restored. To change the name of the database, enter the new name in the **Database** box.
5. In the **Restore to** box, leave the default as **To the last backup taken** or click on **Timeline** to access the **Backup Timeline** dialog box to manually select a point in time to stop the recovery action. See [Backup Timeline](#) for more information on designating a specific point in time.
6. In the **Backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Backups that depend on the restoration of an earlier backup are automatically deselected when the earlier backup is deselected.
For information about the columns in the **Backup sets to restore** grid, see [Restore Database \(General Page\)](#).
7. To specify the new location of the database files, select the **Files** page, and then click **Relocate all files to folder**. Provide a new location for the **Data file folder** and **Log file folder**. For more information about this grid, see [Restore Database \(Files Page\)](#).
8. On the **Options** page, adjust the options if you want. For more information about these options, see [Restore Database \(Options Page\)](#).



Using Transact-SQL

▶ To restore a database to a new location, and optionally rename the database

1. Optionally, determine the logical and physical names of the files in the backup set that contains the full database backup that you want to restore. This statement returns a list of the database and log files contained in the backup set. The basic syntax is as follows:

```
RESTORE FILELISTONLY FROM <backup_device> WITH FILE =  
backup_set_file_number
```

Here, `backup_set_file_number` indicates the position of the backup in the media set. You can obtain the position of a backup set by using the [RESTORE HEADERONLY](#) statement. For more information, see "Specifying a Backup Set"

in [RESTORE Arguments \(Transact-SQL\)](#).

This statement also supports a number of WITH options. For more information, see [RESTORE FILELISTONLY \(Transact-SQL\)](#).

2. Use the [RESTORE DATABASE](#) statement to restore the full database backup. By default, data and log files are restored to their original locations. To relocate a database, use the MOVE option to relocate each of the database files and to avoid collisions with existing files.

The basic Transact-SQL syntax for restoring the database to a new location and a new name is:

```
RESTORE DATABASE new_database_name
FROM backup_device [ ,...n ]
[ WITH
{
    [ RECOVERY | NORECOVERY ]
    [ , ] [ FILE = { backup_set_file_number | @backup_set_file_number } ]
    [ , ] MOVE 'logical_file_name_in_backup' TO 'operating_system_file_name' [ ,...n ]
}
;
```



Note

When preparing to relocate a database on a different disk, you should verify that sufficient space is available and identify any potential collisions with existing files. This involves using a [RESTORE VERIFYONLY](#) statement that specifies the same MOVE parameters that you plan to use in your RESTORE DATABASE statement.

The following table describes arguments of this RESTORE statement in terms of restoring a database to a new location. For more information about these arguments, see [RESTORE \(Transact-SQL\)](#).

new_database_name

The new name for the database.



Note

If you are restoring the database to a different server instance, you can use the original database name instead of a new name.

backup_device [,...n]

Specifies a comma-separated list of from 1 to 64 backup devices from which the database backup is to be restored. You can specify a physical backup device, or you can specify a corresponding logical backup device, if defined. To specify a physical backup device, use the DISK or TAPE option:

{ DISK | TAPE } = *physical_backup_device_name*

For more information, see [Backup Devices](#).

{ RECOVERY | NORECOVERY }

If the database uses the full recovery model, you might need to apply transaction log backups after you restore the database. In this case, specify the NORECOVERY option.

Otherwise, use the RECOVERY option, which is the default.

FILE = { *backup_set_file_number* | @*backup_set_file_number* }

Identifies the backup set to be restored. For example, a *backup_set_file_number* of **1** indicates the first backup set on the backup medium and a *backup_set_file_number* of **2** indicates the second backup set. You can obtain the *backup_set_file_number* of a backup set by using the [RESTORE HEADERONLY](#) statement.

When this option is not specified, the default is to use the first backup set on the backup device.

For more information, see "Specifying a Backup Set," in [RESTORE Arguments \(Transact-SQL\)](#).

MOVE '*logical_file_name_in_backup*' TO '*operating_system_file_name*' [,...n]

Specifies that the data or log file specified by *logical_file_name_in_backup* is to be restored to the location specified by *operating_system_file_name*. Specify a MOVE statement for every logical file you want to restore from the backup set to a new location.

Option	Description
<i>logical_file_name_in_backup</i>	<p>Specifies the logical name of a data or log file in the backup set. The logical file name of a data or log file in a backup set matches its logical name in the database when the backup set was created.</p> <p> Note To obtain a list of the logical files from the backup set, use RESTORE FILELISTONLY.</p>
<i>operating_system_file_name</i>	<p>Specifies a new location for the file specified by</p>

	<p><code>logical_file_name_in_backup</code>. The file will be restored to this location. Optionally, <code>operating_system_file_name</code> specifies a new file name for the restored file. This is necessary if you are creating a copy of an existing database on the same server instance.</p>
n	<p>Is a placeholder indicating that you can specify additional MOVE statements.</p>

Example (Transact-SQL)

This example creates a new database named `MyAdvWorks` by restoring a backup of the sample database, which includes two files: `_Data` and `_Log`. This database uses the simple recovery model. The `AdventureWorks2012` database already exists on the server instance, so the files in the backup must be restored to a new location. The `RESTORE FILELISTONLY` statement is used to determine the number and names of the files in the database being restored. The database backup is the first backup set on the backup device.

Note

The examples of backing up and restoring the transaction log, including point-in-time restores, use the `MyAdvWorks_FullRM` database that is created from `AdventureWorks2012` just like the following `MyAdvWorks` example. However, the resulting `MyAdvWorks_FullRM` database must be changed to use the full recovery model by using the following Transact-SQL statement: `ALTER DATABASE <database_name> SET RECOVERY FULL`.

```
USE master;

GO

-- First determine the number and names of the files in the backup.
-- AdventureWorks2012_Backup is the name of the backup device.
RESTORE FILELISTONLY
    FROM AdventureWorks2012_Backup;
-- Restore the files for MyAdvWorks.
RESTORE DATABASE MyAdvWorks
    FROM AdventureWorks2012_Backup
    WITH RECOVERY,
    MOVE 'AdventureWorks2012_Data' TO 'D:\MyData\MyAdvWorks_Data.mdf',
```

```
MOVE 'AdventureWorks2012_Log' TO 'F:\MyLog\MyAdvWorks_Log.ldf';  
GO
```

For an example of how to create a full database backup of the `AdventureWorks2012` database, see [Create a Full Database Backup](#).



Related Tasks

- [Create a Full Database Backup](#)
- [Restore a Database Backup](#)
- [Back Up a Transaction Log \(SQL Server\)](#)
- [Restore a Transaction Log Backup \(SQL Server\)](#)



See Also

[Manage Metadata When Making a Database Available on Another Server Instance](#)

[RESTORE](#)

[Considerations for Copying Databases with Backup and Restore](#)

Complete Database Restores (Full Recovery Model)

In a complete database restore, the goal is to restore the whole database. The whole database is offline for the duration of the restore. Before any part of the database can come online, all data is recovered to a consistent point in which all parts of the database are at the same point in time and no uncommitted transactions exist.

Under the full recovery model, after you restore your data backup or backups, you must restore all subsequent transaction log backups and then recover the database. You can restore a database to a specific *recovery point* within one of these log backups. The recovery point can be a specific date and time, a marked transaction, or a log sequence number (LSN).

When restoring a database, particularly under the full recovery model or bulk-logged recovery model, you should use a single restore sequence. A *restore sequence* consists of one or more restore operations that move data through one or more of the phases of restore.

noteDXDOC112778PADS

Security Note

We recommend that you do not attach or restore databases from unknown or untrusted sources. These databases could contain malicious code that might execute unintended Transact-SQL code or cause errors by modifying the schema or the physical database structure. Before you use a database from an unknown or untrusted source, run [DBCC CHECKDB](#) on the database on a nonproduction server and also examine the code, such as stored procedures or other user-defined code, in the database.

In this Topic:

- Restoring a Database to the Point of Failure
- Restoring a Database to a Point Within a Log Backup
- Related Tasks



Note

For information about support for backups from earlier versions of SQL Server, see the "Compatibility Support" section of [RESTORE \(Transact-SQL\)](#).

Restoring a Database to the Point of Failure

Typically, recovering a database to the point of failure involves the following basic steps:

1. Back up the active transaction log (known as the tail of the log). This creates a tail-log backup. If the active transaction log is unavailable, all transactions in that part of the log are lost.



Important

Under the bulk-logged recovery model, backing up any log that contains bulk-logged operations requires access to all data files in the database. If the data files cannot be accessed, the transaction log cannot be backed up. In that case, you have to manually redo all changes that were made since the most recent log backup.

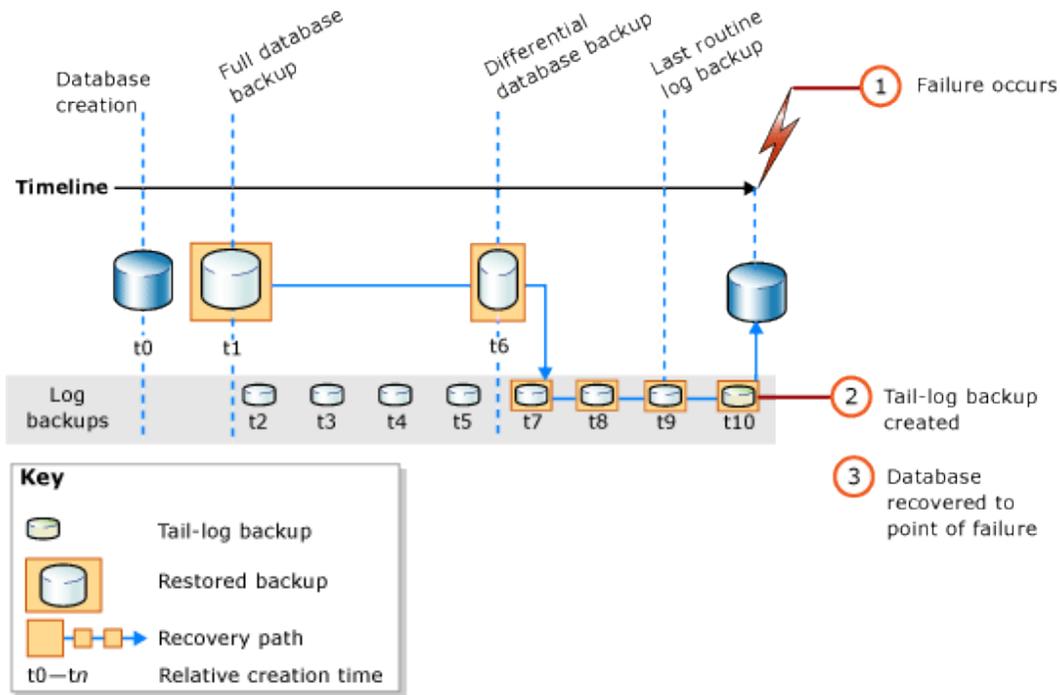
For more information, see [Tail-Log Backups](#).

2. Restore the most recent full database backup without recovering the database (RESTORE DATABASE *database_name* FROM *backup_device* WITH NORECOVERY).
3. If differential backups exist, restore the most recent one without recovering the database (RESTORE DATABASE *database_name* FROM *differential_backup_device* WITH NORECOVERY).

Restoring the most recent differential backup reduces the number of log backups that must be restored.

4. Starting with the first transaction log backup that was created after the backup you just restored, restore the logs in sequence with NORECOVERY.
5. Recover the database (RESTORE DATABASE *database_name* WITH RECOVERY). Alternatively, this step can be combined with restoring the last log backup.

The following illustration shows this restore sequence. After a failure occurs (1), a tail-log backup is created (2). Next, the database is restored to the point of the failure. This involves restoring a database backup, a subsequent differential backup, and every log backup taken after the differential backup, including the tail-log backup.



Note

When you restore a database backup onto a different server instance, see [Copying Databases with Backup and Restore](#).

Basic Transact-SQL RESTORE Syntax

The basic [RESTORE](#) Transact-SQL syntax for the restore sequence in the preceding illustration is as follows:

- RESTORE DATABASE *database* FROM *full database backup* WITH NORECOVERY;
- RESTORE DATABASE *database* FROM *full_differential_backup* WITH NORECOVERY;
- RESTORE LOG *database* FROM *log_backup* WITH NORECOVERY;
Repeat this restore-log step for each additional log backup.
- RESTORE DATABASE *database* WITH RECOVERY;

Example: Recovering to the Point of Failure (Transact-SQL)

The following Transact-SQL example shows the essential options in a restore sequence that restores the database to the point of failure. The example creates a tail-log backup of the database. Next, the example restores a full database backup and log backup and then restores the tail-log backup. The example recovers the database in a separate, final step.



Note

This example uses a database backup and log backup that is created in the "Using Database Backups Under the Full Recovery Model" section in [Full Database Backups](#). Before the database backup, the sample database was set to use the full recovery model.

```
USE master;

--Create tail-log backup.
BACKUP LOG AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdventureWorksFullRM.bak'
WITH NORECOVERY;

GO

--Restore the full database backup (from backup set 1).
RESTORE DATABASE AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdventureWorksFullRM.bak'
WITH FILE=1,
NORECOVERY;

--Restore the regular log backup (from backup set 2).
RESTORE LOG AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdventureWorksFullRM.bak'
WITH FILE=2,
NORECOVERY;

--Restore the tail-log backup (from backup set 3).
RESTORE LOG AdventureWorks2012
FROM DISK = 'Z:\SQLServerBackups\AdventureWorksFullRM.bak'
WITH FILE=3,
NORECOVERY;

GO
```

```
--recover the database:
```

```
RESTORE DATABASE AdventureWorks2012 WITH RECOVERY;  
GO
```

Restoring a Database to a Point Within a Log Backup

Under the full recovery model, a complete database restore can usually be recovered to a point of time, a marked transaction, or an LSN within a log backup. However, under the bulk-logged recovery model, if the log backup contains bulk-logged changes, point-in-time recovery is not possible.

Sample Point-in-Time Restore Scenarios

The following example assumes a mission-critical database system for which a full database backup is created daily at midnight, a differential database backup is created on the hour, Monday through Saturday, and transaction log backups are created every 10 minutes throughout the day. To restore the database to the state it was in at 5:19 A.M. Wednesday, do the following:

1. Restore the full database backup that was created Tuesday at midnight.
2. Restore the differential database backup that was created at 5:00 A.M. on Wednesday.
3. Apply the transaction log backup that was created at 5:10 A.M. on Wednesday.
4. Apply the transaction log backup that was created 5:20 A.M. on Wednesday, specifying that the recovery process applies only to transactions that occurred before 5:19 A.M.

Alternatively, if the database needs to be restored to its state at 3:04 A.M. Thursday, but the differential database backup that was created at 3:00 A.M. Thursday is unavailable, do the following:

1. Restore the database backup that was created Wednesday at midnight.
2. Restore the differential database backup that was created at 2:00 A.M. on Thursday.
3. Apply all the transaction log backups created from 2:10 A.M. to 3:00 A.M. on Thursday.
4. Apply the transaction log backup that was created at 3:10 A.M. on Thursday, stopping the recovery process at 3:04 A.M.



Note

For an example of a point-in-time restore, see [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).

Related Tasks

To restore a full database backup

- [How to: Restore a Database Backup \(SQL Server Management Studio\)](#)
- [How to: Create a New Database From An Existing Database Backup \(SQL Server Management Studio\)](#)

To restore a differential database backup

- [How to: Restore a Full Differential Backup \(SQL Server Management Studio\)](#)

To restore a transaction log backup

- [How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

To restore a backup by using SQL Server Management Objects (SMO)

-

```
M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.M
anagement.Smo.Server)
```

To restore a database to a point within a log backup

- [Recovering to a Specific Point in Time](#)
- [Recovering to a Marked Transaction](#)
- [Recovering to a Log Sequence Number \(LSN\)](#)

See Also

[RESTORE \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[Apply Transaction Log Backups \(SQL Server\)](#)

[sp_addumpdevice \(Transact-SQL\)](#)

[Full Database Backups](#)

[Differential Backups \(SQL Server\)](#)

[Backup Overview \(SQL Server\)](#)

[Restore and Recovery Overview \(SQL Server\)](#)

Restore a Database to the Point of Failure Under the Full Recovery Model (Transact-SQL)

This topic explains how to restore to the point of failure. The topic is relevant only for databases that are using the full or bulk-logged recovery models.

Procedures

▶ To restore to the point of failure

1. Back up the tail of the log by running the following basic [BACKUP](#) statement:

```
BACKUP LOG <database_name> TO <backup_device>
WITH NORECOVERY, NO_TRUNCATE;
```

2. Restore a full database backup by running the following basic [RESTORE DATABASE](#) statement:

```
RESTORE DATABASE <database_name> FROM <backup_device>
WITH NORECOVERY;
```

3. Optionally, restore a differential database backup by running the following basic RESTORE DATABASE statement:

```
RESTORE DATABASE <database_name> FROM <backup_device>
WITH NORECOVERY;
```

4. Apply each transaction log, including the tail-log backup you created in step 1, by specifying WITH NORECOVERY in the RESTORE LOG statement:

```
RESTORE LOG <database_name> FROM <backup_device>
WITH NORECOVERY;
```

5. Recover the database by running the following RESTORE DATABASE statement:

```
RESTORE DATABASE <database_name>
WITH RECOVERY;
```

Example

Description

Before you can run the example, you must complete the following preparations:

1. The default recovery model of the database is the simple recovery model. Because this recovery model does not support restoring to the point of a failure, set to use the full recovery model by running the following [ALTER DATABASE](#) statement:

```
USE master;
GO
ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;
```

2. Create a full database back of the database by using the following BACKUP statement:

```
BACKUP DATABASE AdventureWorks2012 TO DISK =
'C:\AdventureWorks2012_Data.bck';
```

3. Create a routine log backup:

```
BACKUP LOG AdventureWorks2012 TO DISK =
'C:\AdventureWorks2012_Log.bck';
```

The following example restores the backups that are created previously, after creating a tail-log backup of the database. (This step assumes that the log disk can be accessed.)

First, the example creates a tail-log backup of the database that captures the active log and leaves the database in the Restoring state. Then, the example restores the database backup, applies the routine log backup created previously, and applies the tail-log backup. Finally, the example recovers the database in a separate step.



Note

The default behavior is to recover a database as part of the statement that restores the final backup.

Code

```
/* Example of restoring a to the point of failure */
-- Step 1: Create a tail-log backup by using WITH NORECOVERY.
BACKUP LOG AdventureWorks2012
    TO DISK = 'C:\AdventureWorks2012_Log.bck'
    WITH NORECOVERY;
GO
-- Step 2: Restore the full database backup.
RESTORE DATABASE AdventureWorks2012
    FROM DISK = 'C:\AdventureWorks2012_Data.bck'
    WITH NORECOVERY;
GO
-- Step 3: Restore the first transaction log backup.
RESTORE LOG AdventureWorks2012
    FROM DISK = 'C:\AdventureWorks2012_Log.bck'
    WITH NORECOVERY;
GO
-- Step 4: Restore the tail-log backup.
RESTORE LOG AdventureWorks2012
    FROM DISK = 'C:\AdventureWorks2012_Log.bck'
    WITH NORECOVERY;
GO
-- Step 5: Recover the database.
RESTORE DATABASE AdventureWorks2012
```

```
WITH RECOVERY;
```

```
GO
```

See Also

[SQL Server Management Studio Tutorial](#)

[RESTORE \(Transact-SQL\)](#)

File Restores (Simple Recovery Model)

This topic is relevant only for simple-model databases that contain at least one read-only secondary filegroup.

In a file restore, the goal is to restore one or more damaged files without restoring the whole database. Under the simple recovery model, file backups are supported only for read-only files. The primary filegroup and read/write secondary filegroups are always restored together, by restoring a database or partial backup.

The file-restore scenarios are as follows:

- Offline file restore

In an *offline file restore*, the database is offline while damaged files or filegroups are restored. At the end of the restore sequence, the database comes online.

All editions of SQL Server 2012 support offline file restore.

- Online file restore

In an *online file restore*, if database is online at restore time, it remains online during the file restore. However, each filegroup in which a file is being restored is offline during the restore operation. After all the files in an offline filegroup are recovered, the filegroup is automatically brought online.

For information about support for online page and file restore, see [Features Supported by the Editions of SQL Server 2012](#). For more information about online restores, see [Online Restores](#).

Tip

If you want the database to be offline for a file restore, take the database offline before you start the restore sequence by executing the following [ALTER DATABASE](#) statement: ALTER DATABASE `database_name` SET OFFLINE.

In this Topic:

- Overview of File and Filegroup Restore Under the Simple Recovery Model
- Related Tasks

Overview of File and Filegroup Restore Under the Simple Recovery Model

A file restore scenario consists of a single restore sequence that copies, rolls forward, and recovers the appropriate data as follows:

1. Restore each damaged file from its most recent file backup.
2. Restore the most recent differential file backup for each restored file and recover the database.

Transact-SQL Steps for File Restore Sequence (Simple Recovery Model)

This section shows the essential Transact-SQL [RESTORE](#) options for a simple file-restore sequence. Syntax and details that are not relevant to this purpose are omitted.

The restore sequence contains only two Transact-SQL statements. The first statement restores a secondary file, file *A*, which is restored using `WITH NORECOVERY`. The second operation restores two other files, *B* and *C* which are restored using `WITH RECOVERY` from a different backup device:

1.

```
RESTORE DATABASE database FILE = name_of_file_A
FROM file_backup_of_file_A
WITH NORECOVERY;
```
2.

```
RESTORE DATABASE database FILE = name_of_file_B, name_of_file_C
FROM file_backup_of_files_B_and_C
WITH RECOVERY;
```

Examples

- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Offline Restore of Primary and One Other Filegroup](#)

Related Tasks

To restore files and filegroups

- [Restore Files and Filegroups over Existing Files \(Transact-SQL\)](#)
- [Restore Files and Filegroups \(SQL Server Management Studio\)](#)
- [Restore Files and Filegroups \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.Management.Smo.Server) (SMO)

See Also

[Backup and Restore Considerations for Related Features](#)

[Differential Backups \(SQL Server\)](#)

[Full File Backups](#)

[Backup Overview \(SQL Server\)](#)

[Restore and Recovery in SQL Server](#)

[RESTORE \(Transact-SQL\)](#)

[Complete Database Restores \(Simple Recovery Model\)](#)

[Performing Piecemeal Restores](#)

Restore Files and Filegroups over Existing Files

This topic describes how to restore files and filegroups over existing files in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Security
- **To restore files and filegroups over existing files, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The system administrator who is restoring the files and filegroups must be the only person currently using the database to be restored.
- RESTORE is not allowed in an explicit or implicit transaction.
- Under the full or bulk-logged recovery model, before you can restore files, you must back up the active transaction log (known as the tail of the log). For more information, see [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#).
- To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore files and filegroups over existing files

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, expand that instance, and then expand **Databases**.
2. Right-click the database that you want, point to **Tasks**, point to **Restore**, and then click **Files and Filegroups**.
3. On the **General** page, in the **To database** list box, enter the database to restore. You can enter a new database or choose an existing database from the drop-down list. The list includes all databases on the server, excluding the system databases **master** and **tempdb**.
4. To specify the source and location of the backup sets to restore, click one of the following options:
 - **From database**
Enter a database name in the list box. This list contains only databases that have been backed up according to the **msdb** backup history.
 - **From device**
Click the browse button. In the **Specify backup devices** dialog box, select one of the listed device types in the **Backup media type** list box. To select one or more devices for the **Backup media** list box, click **Add**.
After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.
5. In the **Select the backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Any backups that depend on a deselected backup are deselected automatically.

Column head	Values
Restore	The selected check boxes indicate the backup sets to be restored.
Name	The name of the backup set.
File Type	Specifies the type of data in the backup: Data , Log , or Filestream Data . Data that is contained in tables is in Data files. Transaction log data is in Log files. Binary large object (BLOB) data that is stored on the file system is in Filestream Data files.
Type	The type of backup performed: Full , Differential , or Transaction Log .
Server	The name of the Database-Engine instance that performed the backup operation.
File Logical Name	The logical name of the file.
Database	The name of the database involved in the backup operation.
Start Date	The date and time when the backup operation began, presented in the regional setting of the client.
Finish Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.

6. In the **Select a page** pane, click the **Options** page.
7. In the **Restore options** panel, select **Overwrite the existing database (WITH REPLACE)**. The restore operation overwrites any existing databases and their related files, even if another database or file already exists with the same name.
8. Click .



Using Transact-SQL

▶ To restore files and filegroups over existing files

1. Execute the RESTORE DATABASE statement to restore the file and filegroup backup, specifying:
 - The name of the database to restore.
 - The backup device from where the full database backup will be restored.
 - The FILE clause for each file to restore.
 - The FILEGROUP clause for each filegroup to restore.
 - The REPLACE option to specify that each file can be restored over existing files of the same name and location.

⚠ Caution

Use the REPLACE option cautiously. For more information, see .

- The NORECOVERY option. If the files have not been modified after the backup was created, specify the RECOVERY clause.
2. If the files have been modified after the file backup was created, execute the RESTORE LOG statement to apply the transaction log backup, specifying:
 - The name of the database to which the transaction log will be applied.
 - The backup device from where the transaction log backup will be restored.
 - The NORECOVERY clause if you have another transaction log backup to apply after the current one; otherwise, specify the RECOVERY clause.

The transaction log backups, if applied, must cover the time when the files and filegroups were backed up.

Example (Transact-SQL)

The following example restores the files and filegroups for the MyNwind database, and replaces any existing files of the same name. Two transaction logs will also be applied to restore the database to the current time.

```
USE master;
GO
-- Restore the files and filesgroups for MyNwind.
RESTORE DATABASE MyNwind
    FILE = 'MyNwind_data_1',
    FILEGROUP = 'new_customers',
    FILE = 'MyNwind_data_2',
```

```
FILEGROUP = 'first_qtr_sales'  
FROM MyNwind_1  
WITH NORECOVERY,  
REPLACE;  
  
GO  
  
-- Apply the first transaction log backup.  
RESTORE LOG MyNwind  
FROM MyNwind_log1  
WITH NORECOVERY;  
  
GO  
  
-- Apply the last transaction log backup.  
RESTORE LOG MyNwind  
FROM MyNwind_log2  
WITH RECOVERY;  
  
GO
```



See Also

[How to: Restore a Database Backup \(SQL Server Management Studio\)](#)

[RESTORE](#)

[Restore Files and Filegroups](#)

[Considerations for Copying Databases with Backup and Restore](#)

Restore Files to a New Location

This topic describes how to restore files to a new location in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
 - Security
- **To restore files to a new location, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The system administrator restoring the files must be the only person currently using the database to be restored.
- RESTORE is not allowed in an explicit or implicit transaction.
- Under the full or bulk-logged recovery model, before you can restore files, you must back up the active transaction log (known as the tail of the log). For more information, see [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#).
- To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore files to a new location

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, expand that instance, and then expand **Databases**.
2. Right-click the database that you want, point to **Tasks**, point to **Restore**, and then click **Files and Filegroups**.
3. On the **General** page, in the **To database** list box, enter the database to restore. You can enter a new database or choose an existing database from the drop-down list. The list includes all databases on the server, excluding the system

databases **master** and **tempdb**.

4. To specify the source and location of the backup sets to restore, click one of the following options:
 - **From database**

Enter a database name in the list box. This list contains only databases that have been backed up according to the **msdb** backup history.
 - **From device**

Click the browse button. In the **Specify backup devices** dialog box, select one of the listed device types in the **Backup media type** list box. To select one or more devices for the **Backup media** list box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.
5. In the **Select the backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Any backups that depend on a deselected backup are deselected automatically.

Column head	Values
Restore	The selected check boxes indicate the backup sets to be restored.
Name	The name of the backup set.
File Type	Specifies the type of data in the backup: Data , Log , or Filestream Data . Data that is contained in tables is in Data files. Transaction log data is in Log files. Binary large object (BLOB) data that is stored on the file system is in Filestream Data files.
Type	The type of backup performed: Full , Differential , or Transaction Log .
Server	The name of the Database-Engine instance that performed the backup operation.
File Logical Name	The logical name of the file.
Database	The name of the database involved in the backup operation.

Start Date	The date and time when the backup operation began, presented in the regional setting of the client.
Finish Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.

6. In the **Select a page** pane, click the **Options** page.
7. In the **Restore database files as** grid, specify a new location for the file or files that you want to move.

Column head	Values
Original File Name	The full path of a source backup file.
File Type	Specifies the type of data in the backup: Data , Log , or Filestream Data . Data that is contained in tables is in Data files. Transaction log data is in Log files. Binary large object (BLOB) data that is stored on the file system is in Filestream Data files.
Restore As	The full path of the database file to be restored. To specify a new restore file, click the text box and edit the suggested path and file name. Changing the path or file name in the Restore As column is equivalent to using the MOVE option in a Transact-SQL RESTORE statement.

8. Click .



Using Transact-SQL

▶ To restore files to a new location

1. Optionally, execute the RESTORE FILELISTONLY statement to determine the number and names of the files in the full database backup.
2. Execute the RESTORE DATABASE statement to restore the full database backup, specifying:
 - The name of the database to restore.
 - The backup device from where the full database backup will be restored.
 - The MOVE clause for each file to restore to a new location.
 - The NORECOVERY clause.
3. If the files have been modified after the file backup was created, execute the RESTORE LOG statement to apply the transaction log backup, specifying:
 - The name of the database to which the transaction log will be applied.
 - The backup device from where the transaction log backup will be restored.
 - The NORECOVERY clause if you have another transaction log backup to apply after the current one; otherwise, specify the RECOVERY clause.

The transaction log backups, if applied, must cover the time when the files and filegroups were backed up.

Example (Transact-SQL)

This example restores two of the files for the MyNwind database that were originally located on Drive C to new locations on Drive D. Two transaction logs will also be applied to restore the database to the current time. The RESTORE FILELISTONLY statement is used to determine the number and logical and physical names of the files in the database being restored.

```
USE master;
GO
-- First determine the number and names of the files in the backup.
RESTORE FILELISTONLY
    FROM MyNwind_1;
-- Restore the files for MyNwind.
RESTORE DATABASE MyNwind
    FROM MyNwind_1
    WITH NORECOVERY,
    MOVE 'MyNwind_data_1' TO 'D:\MyData\MyNwind_data_1.mdf',
    MOVE 'MyNwind_data_2' TO 'D:\MyData\MyNwind_data_2.ndf';
GO
```

```
-- Apply the first transaction log backup.
RESTORE LOG MyNwind
    FROM MyNwind_log1
    WITH NORECOVERY;
GO

-- Apply the last transaction log backup.
RESTORE LOG MyNwind
    FROM MyNwind_log2
    WITH RECOVERY;
GO

```

See Also

[How to: Restore a Database Backup \(SQL Server Management Studio\)](#)

[RESTORE](#)

[Considerations for Copying Databases with Backup and Restore](#)

[Restore Files and Filegroups](#)

Restore Files and Filegroups

This topic describes how to restore files and filegroups in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
- Security
- **To restore files and filegroups, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Limitations and Restrictions

- The system administrator restoring the files and filegroups must be the only person currently using the database to be restored.
- RESTORE is not allowed in an explicit or implicit transaction.

- Under the simple recovery model, the file must belong to a read-only filegroup.
- Under the full or bulk-logged recovery model, before you can restore files, you must back up the active transaction log (known as the tail of the log). For more information, see [How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#).
- To restore a database that is encrypted, you must have access to the certificate or asymmetric key that was used to encrypt the database. Without the certificate or asymmetric key, the database cannot be restored. As a result, the certificate that is used to encrypt the database encryption key must be retained as long as the backup is needed. For more information, see [SQL Server Certificates and Asymmetric Keys](#).

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

▶ To restore files and filegroups

1. After you connect to the appropriate instance of the SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases**, and then select a system database.
3. Right-click the database, point to **Tasks**, and then click **Restore**.
4. Click **Files and Filegroups**, which opens the **Restore Files and Filegroups** dialog box.
5. On the **General** page, in the **To database** list box, enter the database to restore. You can enter a new database or choose an existing database from the drop-down list. The list includes all databases on the server, excluding the system databases **master** and **tempdb**.

6. To specify the source and location of the backup sets to restore, click one of the following options:
 - **From database**
Enter a database name in the list box. This list contains only databases that have been backed up according to the **msdb** backup history.
 - **From device**
Click the browse button. In the **Specify backup devices** dialog box, select one of the listed device types in the **Backup media type** list box. To select one or more devices for the **Backup media** list box, click **Add**.
After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.
7. In the **Select the backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Any backups that depend on a deselected backup are deselected automatically.

Column head	Values
Restore	The selected check boxes indicate the backup sets to be restored.
Name	The name of the backup set.
File Type	Specifies the type of data in the backup: Data , Log , or Filestream Data . Data that is contained in tables is in Data files. Transaction log data is in Log files. Binary large object (BLOB) data that is stored on the file system is in Filestream Data files.
Type	The type of backup performed: Full , Differential , or Transaction Log .
Server	The name of the Database-Engine instance that performed the backup operation.
File Logical Name	The logical name of the file.
Database	The name of the database involved in the backup operation.
Start Date	The date and time when the backup

	operation began, presented in the regional setting of the client.
Finish Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.

8. To view or select the advanced options, click **Options** in the **Select a page pane**.
9. In the **Restore options** panel, you can choose any of the following options, if appropriate for your situation.

Restore as filegroup

Indicates that an entire filegroup is being restored.

Overwrite the existing database

Specifies that the restore operation should overwrite any existing databases and their related files, even if another database or file already exists with the same name.

Selecting this option is equivalent to using the REPLACE option in a Transact-SQL RESTORE statement.

Prompt before restoring each backup

Asks you for confirmation before restoring each backup set.

This option is particularly useful where you must swap tapes for different media sets, such as when the server has one tape device.

Restrict access to the restored database

Makes the restored database available only to the members of **db_owner**, **dbcreator**, or **sysadmin**.

Selecting this option is synonymous to using the RESTRICTED_USER option in a Transact-SQL RESTORE statement.

10. Optionally, you can restore the database to a new location by specifying a new restore destination for each file in the **Restore database files as** grid.

Column head	Values
Original File Name	The full path of a source backup file.
File Type	Specifies the type of data in the

	backup: Data, Log, or Filestream Data . Data that is contained in tables is in Data files. Transaction log data is in Log files. Binary large object (BLOB) data that is stored on the file system is in Filestream Data files.
Restore As	The full path of the database file to be restored. To specify a new restore file, click the text box and edit the suggested path and file name. Changing the path or file name in the Restore As column is equivalent to using the MOVE option in a Transact-SQL RESTORE statement.

11. The **Recovery state** panel determines the state of the database after the restore operation.

Leave the database ready for use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored. (RESTORE WITH RECOVERY)

Recovers the database. This is the default behavior. Choose this option only if you are restoring all of the necessary backups now. This option is equivalent to specifying WITH RECOVERY in a Transact-SQL RESTORE statement.

Leave the database non-operational, and don't roll back the uncommitted transactions. Additional transaction logs can be restored. (RESTORE WITH NORECOVERY)

Leaves the database in the restoring state. To recover the database, you will need to perform another restore using the preceding RESTORE WITH RECOVERY option (see above). This option is equivalent to specifying WITH NORECOVERY in a Transact-SQL RESTORE statement.

If you select this option, the **Preserve replication settings** option is unavailable.

Leave the database in read-only mode. Roll back the uncommitted transactions, but save the rollback operation in a file so the recovery effects can be undone. (RESTORE WITH STANDBY)

Leaves the database in a standby state. This option is equivalent to specifying WITH STANDBY in a Transact-SQL RESTORE statement.

Choosing this option requires that you specify a standby file.

Rollback undo file

Specify a standby file name in the **Rollback undo file** text box. This option is

required if you leave the database in read-only mode (RESTORE WITH STANDBY).



Using Transact-SQL

▶ To restore files and filegroups

1. Execute the RESTORE DATABASE statement to restore the file and filegroup backup, specifying:
 - The name of the database to restore.
 - The backup device from where the full database backup will be restored.
 - The FILE clause for each file to restore.
 - The FILEGROUP clause for each filegroup to restore.
 - The NORECOVERY clause. If the files have not been modified after the backup was created, specify the RECOVERY clause.
2. If the files have been modified after the file backup was created, execute the RESTORE LOG statement to apply the transaction log backup, specifying:
 - The name of the database to which the transaction log will be applied.
 - The backup device from where the transaction log backup will be restored.
 - The NORECOVERY clause if you have another transaction log backup to apply after the current one; otherwise, specify the RECOVERY clause.

The transaction log backups, if applied, must cover the time when the files and filegroups were backed up until the end of log (unless ALL database files are restored).

Example (Transact-SQL)

This example restores the files and filegroups for the `MyDatabase` database. To restore the database to the current time, two transaction logs are applied.

```
USE master;
GO
-- Restore the files and filesgroups for MyDatabase.
RESTORE DATABASE MyDatabase
    FILE = 'MyDatabase_data_1',
    FILEGROUP = 'new_customers',
    FILE = 'MyDatabase_data_2',
    FILEGROUP = 'first_qtr_sales'
FROM MyDatabase_1
```

```
    WITH NORECOVERY;
GO
-- Apply the first transaction log backup.
RESTORE LOG MyDatabase
    FROM MyDatabase_log1
    WITH NORECOVERY;
GO
-- Apply the last transaction log backup.
RESTORE LOG MyDatabase
    FROM MyDatabase_log2
    WITH RECOVERY;
GO

```

See Also

[How to: Restore a Database Backup \(SQL Server Management Studio\)](#)

[How to: Back Up Database Files and Filegroups \(SQL Server Management Studio\)](#)

[How to: Back Up a Database \(SQL Server Management Studio\)](#)

[How to: Back Up a Transaction Log \(SQL Server Management Studio\)](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

[RESTORE \(Transact-SQL\)](#)

File Restores (Full Recovery Model)

This topic is relevant only for databases that contain multiple files or filegroups under the full or bulk-load recovery model.

In a file restore, the goal is to restore one or more damaged files without restoring the whole database. A file restore scenario consists of a single restore sequence that copies, rolls forward, and recovers the appropriate data

If the filegroup that is being restored is read/write, an unbroken chain of log backups must be applied after the last data or differential backup is restored. This brings the filegroup forward to the log records in the current active log records in the log file. The recovery point is typically near the end of log, but not necessarily.

If the filegroup that is being restored is read-only, usually applying log backups is unnecessary and is skipped. If the backup was taken after the file became read-only, that is the last backup to restore. Roll forward stops at the target point.

The file-restore scenarios are as follows:

- Offline file restore

In an *offline file restore*, the database is offline while damaged files or filegroups are restored. At the end of the restore sequence, the database comes online.

All editions of SQL Server 2012 support offline file restore.

- Online file restore

In an *online file restore*, if database is online at restore time, it remains online during the file restore. However, each filegroup in which a file is being restored is offline during the restore operation. After all the files in an offline filegroup are recovered, the filegroup is automatically brought online.

For information about support for online page and file restore, see [Features Supported by the Editions of SQL Server 2012](#). For more information about online restores, see [Online Restores](#).



Tip

If you want the database to be offline for a file restore, take the database offline before you start the restore sequence by executing the following [ALTER DATABASE](#) statement: ALTER DATABASE `database_name` SET OFFLINE.

In this Topic:

- Restoring Damaged Files from File Backups
- Related Tasks

Restoring Damaged Files from File Backups

1. Before restoring one or more damaged files, attempt to create a tail-log backup.

If the log has been damaged, a tail-log backup cannot be created, and you must restore the whole database.

For information about how to back up a transaction log, see [Creating Transaction Log Backups](#).



Important

For an offline file restore, you must always take a tail-log backup before the file restore. For an online file restore, you must always take the log backup after the file restore. This log backup is necessary to allow for the file to be recovered to a state consistent with the rest of the database.

2. Restore each damaged file from the most recent file backup of that file.
3. Restore the most recent differential file backup, if any, for each restored file.
4. Restore transaction log backups in sequence, starting with the backup that covers the oldest of the restored files and ending with the tail-log backup created in step 1.

You must restore the transaction log backups that were created after the file backups to bring the database to a consistent state. The transaction log backups can be rolled forward quickly, because only the changes that apply to the restored files are applied. Restoring individual files can be better than restoring the whole database, because undamaged files are not copied and then rolled forward. However, the whole chain of log backups still has to be read.

5. Recover the database.



Note

File backups can be used to restore the database to an earlier point in time. To do this, you must restore a complete set of file backups, and then restore transaction log backups in sequence to reach a target point that is after the end of the most recent restored file backup. For more information about point-in-time recovery, see [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).

Transact-SQL Restore Sequence for an Offline File Restore (Full Recovery Model)

A file restore scenario consists of a single restore sequence that copies, rolls forward, and recovers the appropriate data.

This section shows the essential [RESTORE](#) options for a file-restore sequence. Syntax and details that are not relevant to this purpose are omitted.

The following sample restore sequence shows an offline restore of two secondary files, **A** and **B**, using `WITH NORECOVERY`. Next, two log backups are applied with `NORECOVERY`, followed with the tail-log backup, and this is restored using `WITH RECOVERY`.



Note

The following sample restore sequence starts by taking the file offline and then creates a tail-log backup.

```
--Take the file offline.  
ALTER DATABASE database_name MODIFY FILE SET OFFLINE;  
  
-- Back up the currently active transaction log.  
BACKUP LOG database_name  
    TO <tail_log_backup>  
    WITH NORECOVERY;  
  
GO  
  
-- Restore the files.  
RESTORE DATABASE database_name FILE=name  
    FROM <file_backup_of_file_A>
```

```

WITH NORECOVERY;
RESTORE DATABASE database_name FILE=<name> .....
FROM <file_backup_of_file_B>
WITH NORECOVERY;
-- Restore the log backups.
RESTORE LOG database_name FROM <log_backup>
WITH NORECOVERY;
RESTORE LOG database_name FROM <log_backup>
WITH NORECOVERY;
RESTORE LOG database_name FROM <tail_log_backup>
WITH RECOVERY;

```

Examples

- [Example: Online Restore of a Read-Write File](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)
- [Example: Offline Restore of Primary and One Other Filegroup](#)

Related Tasks

To restore files and filegroups

- [Restore Files to a New Location \(Transact-SQL\)](#)
- [Restore Files and Filegroups \(SQL Server Management Studio\)](#)
-

M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.Management.Smo.Server) (SMO)

See Also

[Backup and Restore Considerations for Related Features](#)

[Differential Backups \(SQL Server\)](#)

[Full File Backups](#)

[Backup Overview \(SQL Server\)](#)

[Restore and Recovery in SQL Server](#)

[RESTORE \(Transact-SQL\)](#)

[Complete Database Restores \(Simple Recovery Model\)](#)

[Performing Piecemeal Restores](#)

Apply Transaction Log Backups

The topic is relevant only for the full recovery model or bulk-logged recovery model.

This topic describes applying transaction log backups as part of restoring a SQL Server database.

In this Topic:

- Requirements for Restoring Transaction Log Backups
- Recovery and Transaction Logs
- Using Log Backups to Restore to the Point of Failure
- Related Tasks

Requirements for Restoring Transaction Log Backups

To apply a transaction log backup, the following requirements must be met:

- **Enough Log Backups for a Restore Sequence** : You must have enough log records backed up to complete a restore sequence. The necessary log backups, including the tail-log backup where required, must be available before the start of the restore sequence.
- **Correct restore order**: The immediately previous full database backup or differential database backup must be restored first. Then, all transaction logs that are created after that full or differential database backup must be restored in chronological order. If a transaction log backup in this log chain is lost or damaged, you can restore only transaction logs before the missing transaction log.
- **Database not yet recovered**: The database cannot be recovered until after the final transaction log has been applied. If you recover the database after restoring one of the intermediate transaction log backups, that before the end of the log chain, you cannot restore the database past that point without restarting the complete restore sequence, starting with the full database backup.



Tip

A best practice is to restore all the log backups (`RESTORE LOG database_name WITH NORECOVERY`). Then, after restoring the last log backup, recover the database in a separate operation (`RESTORE DATABASE database_name WITH RECOVERY`).

Recovery and Transaction Logs

When you finish the restore operation and recover the database, recovery rolls back all incomplete transactions. This is known as the *undo phase*. Rolling back is required to restore the integrity of the database. After rollback, the database goes online, and no more transaction log backups can be applied to the database.

For example, a series of transaction log backups contain a long-running transaction. The start of the transaction is recorded in the first transaction log backup, but the end of the transaction is recorded in the second transaction log backup. There is no record of a commit or rollback operation in the first transaction log backup. If a recovery operation runs when the first transaction log backup is applied, the long-running transaction is treated as incomplete, and data modifications recorded in the first transaction log backup for the transaction are rolled back. SQL Server does not allow for the second transaction log backup to be applied after this point.



Note

In some circumstances, you can explicitly add a file during log restore.



Using Log Backups to Restore to the Point of Failure

Assume the following sequence of events.

Time	Event
8:00 A.M.	Back up database to create a full database backup.
Noon	Back up transaction log.
4:00 P.M.	Back up transaction log.
6:00 P.M.	Back up database to create a full database backup.
8:00 P.M.	Back up transaction log.
9:45 P.M.	Failure occurs.



Note

For an explanation of this example sequence of backups, see [Creating Transaction Log Backups](#).

To restore the database to its state at 9:45 P.M. (the point of failure), either of the following alternative procedures can be used:

Alternative 1: Restore the database by using the most recent full database backup

1. Create a tail-log backup of the currently active transaction log as of the point of failure.
2. Do not restore the 8:00 A.M. full database backup. Instead, restore the more recent 6:00 P.M. full database backup, and then apply the 8:00 P.M. log backup and the tail-log backup.

Alternative 2: Restore the database by using an earlier full database backup

Note

This alternative process is useful if a problem prevents you from using the 6:00 P.M. full database backup. This process takes longer than restoring from the 6:00 P.M. full database backup.

1. Create a tail-log backup of the currently active transaction log as of the point of failure.
2. Restore the 8:00 A.M. full database backup, and then restore all four transaction log backups in sequence. This rolls forward all completed transactions up to 9:45 P.M. This alternative points out the redundant security offered by maintaining a chain of transaction log backups across a series of full database backups.

Note

In some cases, you can also use transaction logs to restore a database to a specific point in time. For more information, [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).



Related Tasks

To apply a transaction log backup

- [Restore a Transaction Log Backup \(SQL Server\)](#)

To restore to your recovery point

- [Restore to the Point of Failure \(Transact-SQL\)](#)
- [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#)



`M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.Management.Smo.Server)` (SMO)

- [Recovering to a Marked Transaction](#)
- [Recovering to a Log Sequence Number \(LSN\)](#)

To recover a database after restoring backups using WITH NORECOVERY

- [Recover a Database Without Restoring Data \(Transact-SQL\)](#)



See Also

[The Transaction Log \(SQL Server\)](#)

Restore a Transaction Log Backup

This topic describes how to restore a transaction log backup in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Prerequisites
 - Security
- **To restore a transaction log backup, using:**
 - SQL Server Management Studio
 - Transact-SQL
- Related Tasks

Before You Begin

Prerequisites

- Backups must be restored in the order in which they were created. Before you can restore a particular transaction log backup, you must first restore the following previous backups without rolling back uncommitted transactions, that is WITH NORECOVERY:
 - The full database backup and the last differential backup, if any, taken before the particular transaction log backup. Before the most recent full or differential database backup was created, the database must have been using the full recovery model or bulk-logged recovery model.
 - All transaction log backups taken after the full database backup or the differential backup (if you restore one) and before the particular transaction log backup. Log backups must be applied in the sequence in which they were created, without any gaps in the log chain.

For more information about transaction log backups, see [Transaction Log Backups \(SQL Server\)](#) and [Apply Transaction Log Backups](#).

Security

Permissions

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when

RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

Warning

The normal process of a restore is to select the log backups in the **Restore Database** dialog box along with the data and differential backups.

To restore a transaction log backup

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, point to **Restore**, and then click **Transaction Log**, which opens the **Restore Transaction Log** dialog box.

Note

If **Transaction Log** is grayed out, you may need to restore a full or differential backup first. Use the **Database** backup dialog box.

4. On the **General** page, in the **Database** list box, select the name of a database. Only databases in the restoring state are listed.
5. To specify the source and location of the backup sets to restore, click one of the following options:

- **From previous backups of database**

Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.

- **From file or tape**

Click the browse (...) button to open the **Select backup devices** dialog box.

In the **Backup media type** box, select one of the listed device types. To select one or more devices for the **Backup media** box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.

6. In the **Select the transaction log backups to restore** grid, select the backups to restore. This grid lists the transaction log backups available for the selected database. A log backup is available only if its **First LSN** greater than the **Last LSN** of the database. Log backups are listed in the order of the log sequence numbers (LSN) they contain, and they must be restored in this order.

The following table lists the column headers of the grid and describes their

values.

Header	Value
Restore	Selected check boxes indicate the backup sets to be restored.
Name	Name of the backup set.
Component	Backed-up component: Database , File , or <blank> (for transaction logs).
Database	Name of the database involved in the backup operation.
Start Date	Date and time when the backup operation began, presented in the regional setting of the client.
Finish Date	Date and time when the backup operation finished, presented in the regional setting of the client.
First LSN	Log sequence number of the first transaction in the backup set. Blank for file backups.
Last LSN	Log sequence number of the last transaction in the backup set. Blank for file backups.
Checkpoint LSN	Log sequence number of the most recent checkpoint at the time the backup was created.
Full LSN	Log sequence number of the most recent full database backup.
Server	Name of the Database Engine instance that performed the backup operation.
User Name	Name of the user who performed the backup operation.
Size	Size of the backup set in bytes.
Position	Position of the backup set in the volume.
Expiration	Date and time the backup set expires.

7. Select one of the following:

- **Point in time**

Either retain the default (**Most recent possible**) or select a specific date and time by clicking the browse button, which opens the **Point in Time Restore** dialog box.

- **Marked transaction**

Restore the database to a previously marked transaction. Selecting this option launches the **Select Marked Transaction** dialog box, which displays a grid listing the marked transactions available in the selected transaction log backups.

By default, the restore is up to, but excluding, the marked transaction. To restore the marked transaction also, select **Include marked transaction**.

The following table lists the column headers of the grid and describes their values.

Header	Value
<blank>	Displays a checkbox for selecting the mark.
Transaction Mark	Name of the marked transaction specified by the user when the transaction was committed.
Date	Date and time of the transaction when it was committed. Transaction date and time are displayed as recorded in the msdb gmarkhistory table, not in the client computer's date and time.
Description	Description of marked transaction specified by the user when the transaction was committed (if any).
LSN	Log sequence number of the marked transaction.
Database	Name of the database where the marked transaction was committed.
User Name	Name of the database user who committed the marked transaction.

8. To view or select the advanced options, click **Options** in the **Select a page** pane.
9. In the **Restore options** section, the choices are:
 - **Preserve the replication settings (WITH KEEP_REPLICATION)**

Preserves the replication settings when restoring a published database to a server other than the server where the database was created.

This option is available only with the **Leave the database ready for use by rolling back the uncommitted transactions...** option (described later), which is equivalent to restoring a backup with the **RECOVERY** option.

Checking this option is equivalent to using the **KEEP_REPLICATION** option in a Transact-SQL **RESTORE** statement.
 - **Prompt before restoring each backup**

Before restoring each backup set (after the first), this option brings up the **Continue with Restore** dialog box, which asks you to indicate whether you want to continue the restore sequence. This dialog displays the name of the next media set (if available), the backup set name, and backup set description.

This option is particularly useful when you must swap tapes for different media sets. For example, you can use it when the server has only one tape device. Wait until you are ready to proceed before clicking **OK**.

Clicking **No** leaves the database in the restoring state. At your convenience, you can continue the restore sequence after the last restore that completed. If the next backup is a data or differential backup, use the **Restore Database** task again. If the next backup is a log backup, use the **Restore Transaction Log** task.
 - **Restrict access to the restored database (WITH RESTRICTED_USER)**

Makes the restored database available only to the members of **db_owner**, **dbcreator**, or **sysadmin**.

Checking this option is synonymous to using the **RESTRICTED_USER** option in a Transact-SQL **RESTORE** statement.
10. For the **Recovery state** options, specify the state of the database after the restore operation.
 - **Leave the database ready for use by rolling back uncommitted transactions. Additional transaction logs cannot be restored. (RESTORE WITH RECOVERY)**

Recovers the database. This option is equivalent to the **RECOVERY** option in a Transact-SQL **RESTORE** statement.

Choose this option only if you have no log files you want to restore.
 - **Leave the database non-operational, and do not roll back uncommitted transactions. Additional transaction logs can be restored. (RESTORE**

WITH NORECOVERY)

Leaves the database unrecovered, in the **RESTORING** state. This option is equivalent to using the **NORECOVERY** option in a Transact-SQL **RESTORE** statement.

When you choose this option, the **Preserve replication settings** option is unavailable.

Important

For a mirror or secondary database, always select this option.

- **Leave the database in read-only mode. Undo uncommitted transactions, but save the undo actions in a file so that recovery effects can be reversed. (RESTORE WITH STANDBY)**

Leaves the database in a standby state. This option is equivalent to using the **STANDBY** option in a Transact-SQL **RESTORE** statement.

Choosing this option requires that you specify a standby file.

11. Optionally, specify a standby file name in the **Standby file** text box. This option is required if you leave the database in read-only mode. You can browse for the standby file or type its pathname in the text box.



Using Transact-SQL

Important

We recommend that you always explicitly specify either **WITH NORECOVERY** or **WITH RECOVERY** in every **RESTORE** statement to eliminate ambiguity. This is particularly important when writing scripts.

To restore a transaction log backup

1. Execute the **RESTORE LOG** statement to apply the transaction log backup, specifying:
 - The name of the database to which the transaction log will be applied.
 - The backup device where the transaction log backup will be restored from.
 - The **NORECOVERY** clause.

The basic syntax for this statement is as follows:

```
RESTORE LOG database_name FROM <backup_device> WITH NORECOVERY.
```

Where *database_name* is the name of database and <backup_device> is the name of the device that contains the log backup being restored.

2. Repeat step 1 for each transaction log backup you have to apply.
3. After restoring the last backup in your restore sequence, to recover the database use one of the following statements:

Recover the database as part of the last RESTORE LOG statement:

```
RESTORE LOG <database_name> FROM <backup_device> WITH  
RECOVERY;  
  
GO
```

Wait to recover the database by using a separate RESTORE DATABASE statement:

```
RESTORE LOG <database_name> FROM <backup_device> WITH  
NORECOVERY;  
  
RESTORE DATABASE <database_name> WITH RECOVERY;  
  
GO
```

Waiting to recover the database gives you the opportunity to verify that you have restored all of the necessary log backups. This approach is often advisable when you are performing a point-in-time restore.

Important

If you are creating a mirror database, omit the recovery step. A mirror database must remain in the RESTORING state.

Examples (Transact-SQL)

By default, the `AdventureWorks2012` database uses the simple recovery model. The following examples require modifying the database to use the full recovery model, as follows:

```
ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;
```

A. Applying a single transaction log backup

The following example starts by restoring the `AdventureWorks2012` database by using a full database backup that resides on a backup device named `AdventureWorks2012_1`. The example then applies the first transaction log backup that resides on a backup device named `AdventureWorks2012_log`. Finally, the example recovers the database.

```
RESTORE DATABASE AdventureWorks2012  
  
FROM AdventureWorks2012_1  
  
WITH NORECOVERY;  
  
GO  
  
RESTORE LOG AdventureWorks2012  
  
FROM AdventureWorks2012_log  
  
WITH FILE = 1,  
  
WITH NORECOVERY;  
  
GO  
  
RESTORE DATABASE AdventureWorks2012
```

```
WITH RECOVERY;
```

```
GO
```

B. Applying multiple transaction log backups

The following example starts by restoring the `AdventureWorks2012` database by using a full database backup that resides on a backup device named `AdventureWorks2012_1`. The example then applies, one by one, the first three transaction log backups that reside on a backup device named `AdventureWorks2012_log`. Finally, the example recovers the database.

```
RESTORE DATABASE AdventureWorks2012
```

```
FROM AdventureWorks2012_1
```

```
WITH NORECOVERY;
```

```
GO
```

```
RESTORE LOG AdventureWorks2012
```

```
FROM AdventureWorks2012_log
```

```
WITH FILE = 1,
```

```
NORECOVERY;
```

```
GO
```

```
RESTORE LOG AdventureWorks2012
```

```
FROM AdventureWorks2012_log
```

```
WITH FILE = 2,
```

```
WITH NORECOVERY;
```

```
GO
```

```
RESTORE LOG AdventureWorks2012
```

```
FROM AdventureWorks2012_log
```

```
WITH FILE = 3,
```

```
WITH NORECOVERY;
```

```
GO
```

```
RESTORE DATABASE AdventureWorks2012
```

```
WITH RECOVERY;
```

```
GO
```



Related Tasks

- [Back Up a Transaction Log \(SQL Server\)](#)
- [Restore a Full Backup \(SQL Server Management Studio\)](#)
- [Restore to the Point of Failure \(Transact-SQL\)](#)

- [Restore to a Point in Time \(Transact-SQL\)](#)
- [Restore a Database to a Marked Transaction \(SQL Server Management Studio\)](#)



See Also

[RESTORE](#)

[Apply Transaction Log Backups \(SQL Server\)](#)

Restore a SQL Server Database to a Point in Time (Full Recovery Model)

This topic describes how to restore a database to a point in time in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. This topic is relevant only for SQL Server databases that use the full or bulk-logged recovery models.



Important

Under the bulk-logged recovery model, if a log backup contains bulk-logged changes, point-in-time recovery is not possible to a point within that backup. The database must be recovered to the end of the transaction log backup.

- **Before you begin:**

Recommendations

Security

- **To restore a SQL Server database to a point in time, using:**

SQL Server Management Studio

Transact-SQL

Before You Begin

Recommendations

- Use STANDBY to find unknown point in time.
- Specify the point in time early in a restore sequence

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.

Using SQL Server Management Studio

To restore a database to a point in time

1. In Object Explorer, connect to the appropriate instance of the SQL Server Database Engine, and expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases**, and then select a system database.
3. Right-click the database, point to **Tasks**, point to **Restore**, and then click **Database**.
4. On the **General** page, use the **Source** section to specify the source and location of the backup sets to restore. Select one of the following options:

- **Database**

Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.



Note

If the backup is taken from a different server, the destination server will not have the backup history information for the specified database. In this case, select **Device** to manually specify the file or device to restore.

- **Device**

Click the browse (...) button to open the **Select backup devices** dialog box. In the **Backup media type** box, select one of the listed device types. To select one or more devices for the **Backup media** box, click **Add**.

After you add the devices you want to the **Backup media** list box, click **OK** to return to the **General** page.

In the **Source: Device: Database** list box, select the name of the database which should be restored.

Note This list is only available when **Device** is selected. Only databases that have backups on the selected device will be available.

5. In the **Destination** section, the **Database** box is automatically populated with the name of the database to be restored. To change the name of the database, enter the new name in the **Database** box.
6. Click **Timeline** to access the **Backup Timeline** dialog box.
7. In the **Restore to** section, click **Specific date and time**.

8. Use either the **Date** and **Time** boxes or the slider bar to specify a specific date and time to where the restore should stop. Click .



Note

Use the **Timeline Interval** box to change the amount of time displayed on the timeline.

9. After you have specified a specific point in time, only the backups that are required to restore to that point in time are selected in the **Restore** column of the **Backupsets to restore** grid. These selected backups make up the recommended restore plan for your point-in-time restore. You should use only the selected backups for your point-in-time restore operation.

For information about the columns in the **Backup sets to restore** grid, see [SQL Server Management Studio Tutorial](#).

10. On the **Options** page, in the **Restore options** panel, you can select any of the following options, if appropriate for your situation:

- **Overwrite the existing database (WITH REPLACE)**
- **Preserve the replication settings (WITH KEEP_REPLICATION)**
- **Restrict access to the restored database (WITH RESTRICTED_USER)**

For more information about these options, see [Restore Database \(Options Page\)](#).

11. Select an option for the **Recovery state** box. This box determines the state of the database after the restore operation.

- **RESTORE WITH RECOVERY** is the default behavior which leaves the database ready for use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored. Select this option if you are restoring all of the necessary backups now.
- **RESTORE WITH NORECOVERY** which leaves the database non-operational, and does not roll back the uncommitted transactions. Additional transaction logs can be restored. The database cannot be used until it is recovered.
- **RESTORE WITH STANDBY** which leaves the database in read-only mode. It undoes uncommitted transactions, but saves the undo actions in a standby file so that recovery effects can be reverted.

For descriptions of the options, see [Restore Database \(Options Page\)](#).

12. **Take tail-log backup before restore** will be selected if it is necessary for the point in time that you have selected. You do not need to modify this setting, but you can choose to backup the tail of the log even if it is not required.

13. Restore operations may fail if there are active connections to the database. Check the **Close existing connections option** to ensure that all active connections between Management Studio and the database are closed. This check box sets the database to single user mode before performing the restore operations, and sets the database to multi-user mode when complete.

14. Select **Prompt before restoring each backup** if you wish to be prompted between each restore operation. This is not usually necessary unless the database is large and you wish to monitor the status of the restore operation.

Using Transact-SQL

Before you begin

A specified time is always restored from a log backup. In every RESTORE LOG statement of the restore sequence, you must specify your target time or transaction in an identical STOPAT clause. As a prerequisite to a point-in-time restore, you must first restore a full database backup whose end point is earlier than your target restore time. That full database backup can be older than the most recent full database backup as long as you then restore every subsequent log backup, up to and including the log backup that contains your target point in time.

To help you identify which database backup to restore, you can optionally specify your WITH STOPAT clause in your RESTORE DATABASE statement to raise an error if a data backup is too recent for the specified target time. The complete data backup is always restored, even if it contains the target time.

Basic Transact-SQL syntax

```
RESTORE LOG database_name FROM <backup_device> WITH STOPAT = time,  
RECOVERY...
```

The recovery point is the latest transaction commit that occurred at or before the **datetime** value that is specified by *time*.

To restore only the modifications that were made before a specific point in time, specify WITH STOPAT = *time* for each backup you restore. This makes sure that you do not go past the target time.

To restore a database to a point in time

Note

For an example of this procedure, see Example (Transact-SQL), later in this section.

1. Connect to server instance on which you want to restore the database.
2. Execute the RESTORE DATABASE statement using the NORECOVERY option.

Note

If a partial restore sequence excludes any [FILESTREAM](#) filegroup, point-in-time restore is not supported. You can force the restore sequence to continue. However the FILESTREAM filegroups that are omitted from your RESTORE statement can never be restored. To force a point-in-time restore, specify the CONTINUE_AFTER_ERROR option together with the STOPAT, STOPATMARK, or STOPBEFOREMARK option, which you must also specify in

your subsequent RESTORE LOG statements. If you specify CONTINUE_AFTER_ERROR, the partial restore sequence succeeds and the FILESTREAM filegroup becomes unrecoverable.

3. Restore the last differential database backup, if any, without recovering the database (RESTORE DATABASE *database_name* FROM *backup_device* WITH NORECOVERY).
4. Apply each transaction log backup in the same sequence in which they were created, specifying the time at which you intend to stop restoring log (RESTORE DATABASE *database_name* FROM <*backup_device*> WITH STOPAT= *time*, RECOVERY).



Note

The RECOVERY and STOPAT options. If the transaction log backup does not contain the requested time (for example, if the time specified is beyond the end of the time covered by the transaction log), a warning is generated and the database remains unrecovered.

Example (Transact-SQL)

The following example restores a database to its state as of 12:00 AM on April 15, 2020 and shows a restore operation that involves multiple log backups. On the backup device, AdventureWorksBackups, the full database backup to be restored is the third backup set on the device (FILE = 3), the first log backup is the fourth backup set (FILE = 4), and the second log backup is the fifth backup set (FILE = 5).



Important

The database uses the simple recovery model. To permit log backups, before taking a full database backup, the database was set to use the full recovery model, using ALTER DATABASE AdventureWorks SET RECOVERY FULL.

```
RESTORE DATABASE AdventureWorks
FROM AdventureWorksBackups
WITH FILE=3, NORECOVERY;
```

```
RESTORE LOG AdventureWorks
FROM AdventureWorksBackups
WITH FILE=4, NORECOVERY, STOPAT = 'Apr 15, 2020 12:00 AM';
```

```
RESTORE LOG AdventureWorks
FROM AdventureWorksBackups
WITH FILE=5, NORECOVERY, STOPAT = 'Apr 15, 2020 12:00 AM';
```

```
RESTORE DATABASE AdventureWorks WITH RECOVERY;
```

```
GO
```

Related Tasks

- [Restore a Database Backup \(SQL Server Management Studio\)](#)
- [Back Up a Transaction Log \(SQL Server Management Studio\)](#)
- [How to: Restore to the Point of Failure Under the Full Recovery Model \(Transact-SQL\)](#)
- [Restore a Database to a Marked Transaction \(SQL Server\)](#)
- [Recover to a Log Sequence Number \(SQL Server\)](#)
- **P:Microsoft.SqlServer.Management.Smo.Restore.ToPointInTime** (SMO)

See Also

[backupset \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[RESTORE HEADERONLY \(Transact-SQL\)](#)

Recovery of Related Databases That Contain Marked Transaction

This topic is relevant only for databases that contain marked transactions and that use the full or bulk-logged recovery models.

For information about the requirements for restoring to a specific recovery point, see [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#).

SQL Server supports inserting named marks into the transaction log to allow recovery to that specific mark. Log marks are transaction specific and are inserted only if their associated transaction commits. As a result, marks can be tied to specific work, and you can recover to a point that includes or excludes this work.

Before you insert named marks into the transaction log, consider the following:

- Because transaction marks consume log space, use them only for transactions that play a significant role in the database recovery strategy.
- After a marked transaction commits, a row is inserted in the [logmarkhistory](#) table in **msdb**.
- If a marked transaction spans multiple databases on the same database server or on different servers, the marks must be recorded in the logs of all the affected databases. For more information, see [Backup and Recovery of Related Databases](#).

 **Note**

For information about how to mark transactions, see [Ensuring Recoverability of Related Databases \(Using Marked Transactions\)](#).

Transact-SQL Syntax for Inserting Named Marks into a Transaction Log

To insert marks into the transaction logs, use the [BEGIN TRANSACTION](#) statement and the `WITH MARK [description]` clause. The mark is named the same as the transaction. The optional *description* is a textual description of the mark, not the mark name. For example, the name of both the transaction and the mark that is created in the following `BEGIN TRANSACTION` statement is `Tx1`:

```
BEGIN TRANSACTION Tx1 WITH MARK 'not the mark name, just a description'
```

The transaction log records the mark name (transaction name), description, database, user, **datetime** information, and the log sequence number (LSN). The **datetime** information is used with the mark name to uniquely identify the mark.

For information about how to insert a mark into a transaction that spans multiple databases, see [Ensuring Recoverability of Related Databases](#).

Transact-SQL Syntax for Recovering to a Mark

When you target a marked transaction by using a [RESTORE LOG](#) statement, you can use one of the following clauses to stop at or immediately before the mark:

- Use the `WITH STOPATMARK = '<mark_name>'` clause to specify that the marked transaction is the recovery point.
STOPATMARK rolls forward to the mark and includes the marked transaction in the roll forward.
- Use the `WITH STOPBEFOREMARK = '<mark_name>'` clause to specify that the log record that is immediately before the mark is the recovery point.
STOPBEFOREMARK rolls forward to the mark and excludes the marked transaction from the roll forward.

The STOPATMARK and STOPBEFOREMARK options both support an optional AFTER *datetime* clause. When *datetime* is used, mark names do not have to be unique.

If AFTER *datetime* is omitted, roll forward stops at the first mark that has the specified name. If AFTER *datetime* is specified, roll forward stops at the first mark that has the specified name, exactly at or after *datetime*.

Note

As in all point-in-time restore operations, recovering to a mark is disallowed when the database is undergoing operations that are bulk-logged.

To restore to a marked transaction

[How to: Restore a Marked Transaction \(SQL Server Management Studio\)](#)

[RESTORE \(Transact-SQL\)](#)

Preparing the Log Backups

For this example, an appropriate backup strategy for these related databases would be the following:

1. Use the full recovery model for both databases.
2. Create a full backup of each database.
The databases can be backed up sequentially or simultaneously.
3. Before backing up the transaction log, mark a transaction that executes in all databases. For information about how to create the marked transactions, see [Ensuring Recoverability of Related Databases \(Using Marked Transactions\)](#).
4. Back up the transaction log on each database.

Recovering the Database to a Marked Transaction

To restore the backup

1. Create tail-log backups of the undamaged databases, if possible.
2. Restore the most recent full database backup of each database.
3. Identify the most recent marked transaction that is available in all of the transaction log backups. This information is stored in the **logmarkhistory** table in the **msdb** database on each server.
4. Identify the log backups for all related databases that contain this mark.
5. Restore each log backup, stopping at the marked transaction.
6. Recover each database.

See Also

[BEGIN TRANSACTION \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[Applying Transaction Log Backups](#)

[Ensuring Recoverability of Related Databases \(Using Marked Transactions\)](#)

[Restore and Recovery Overview \(SQL Server\)](#)

[Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#)

[Plan and Perform Restore Sequences \(Full Recovery Model\)](#)

Use Marked Transactions to Recover Related Databases Consistently (Full Recovery Model)

This topic is relevant only for SQL Server databases that are using the full or bulk-logged recovery models.

When you make related updates to two or more databases, *related databases*, you can use transaction marks to recover them to a logically consistent point. However, this recovery loses any transaction that is committed after the mark that was used as the

recovery point. Marking transactions is suitable only when you are testing related databases or when you are willing to lose recently committed transactions.

Routinely marking related transactions in every related database establishes a series of common recovery points in the databases. The transaction marks are recorded in the transaction log and included in log backups. In the event of a disaster, you can restore each of the databases to the same transaction mark to recover them to a consistent point.



Note

Log backups on the different databases can be created independently of each other and do not have to be simultaneous.

Recovering related databases in the following scenarios requires that you have already marked transactions in every related database:

- One or more transaction logs are destroyed. You have to restore the set of databases to a consistent state at the time of your last log backup.
- You have to restore the entire set of databases to a mutually consistent state at some earlier point in time.



Important

You can recover related databases only to a marked transaction, not to a specific point in time.

For information about how to create marking transactions, see "Creating the Marked Transactions," later in this topic.

Typical Scenario for Using Marked Transactions

A typical scenario for using marked transactions includes the following steps:

1. Create a full or differential database backup of each of the related databases.
2. Mark a transaction block in all the databases.
3. Back up the transaction log for all the databases.
4. Restore database backups WITH NORECOVERY.
5. Restore logs WITH STOPATMARK.

Considerations for Using Marked Transactions

Before inserting named marks into the transaction log, consider the following:

- Because transaction marks consume log space, use them only for transactions that play a significant role in the database recovery strategy.
- After a marked transaction commits, a row is inserted in the [logmarkhistory](#) table in **msdb**.
- If a marked transaction spans multiple databases on the same database server or on different servers, the marks must be recorded in the logs of all the affected databases.

Creating the Marked Transactions

To create a marked transaction, use the [BEGIN TRANSACTION](#) statement and the WITH MARK [*description*] clause. The optional *description* is a textual description of the mark. A mark name for the transaction is required. A mark name can be reused. The transaction log records the mark name, description, database, user, datetime information, and the log sequence number (LSN). The datetime information is used along with the mark name to uniquely identify the mark.

To create marked transactions in a set of databases:

1. Name the transaction in the BEGIN TRAN statement and use the WITH MARK clause
You can nest the statement BEGIN TRAN *new_mark_name* WITH MARK within an existing transaction. The value of *new_mark_name* is the mark name for the transaction, even if the transaction possesses a transaction name.



Note

If you issue a second nested BEGIN TRAN...WITH MARK, that statement is skipped but causes a warning message.

2. Run an update against all of the databases in the set.
The mark for a specific transaction is inserted into transaction logs only on the server instance where the BEGIN TRAN...WITH MARK statement is executed. The transaction mark is placed in the transaction log of every database updated by the marked transaction on that server instance. If the databases reside on different server instances, identical marks must be created on each of the server instances.

Examples

The following example restores the transaction log to the mark in the marked transaction named ListPriceUpdate.

```
USE AdventureWorks
GO
BEGIN TRANSACTION ListPriceUpdate
    WITH MARK 'UPDATE Product list prices';
GO

UPDATE Production.Product
    SET ListPrice = ListPrice * 1.10
    WHERE ProductNumber LIKE 'BK-%';
GO

COMMIT TRANSACTION ListPriceUpdate;
GO
```

```
-- Time passes. Regular database
-- and log backups are taken.
-- An error occurs in the database.
USE master
GO
```

```
RESTORE DATABASE AdventureWorks
FROM AdventureWorksBackups
WITH FILE = 3, NORECOVERY;
GO
```

```
RESTORE LOG AdventureWorks
    FROM AdventureWorksBackups
    WITH FILE = 4,
    RECOVERY,
    STOPATMARK = 'ListPriceUpdate';
```

Forcing a Mark to Spread to Other Servers

A transaction mark name is not automatically distributed to another server as the transaction spreads there. To force the mark to spread to the other servers, a stored procedure must be written that contains a `BEGIN TRAN name WITH MARK` statement. That stored procedure must then be executed on the remote server under the scope of the transaction in the originating server.

For example, consider a partitioned database that exists on multiple instances of SQL Server. On each instance is a database named `coyote`. First, in every database, create a stored procedure, for example, `sp_SetMark`.

```
CREATE PROCEDURE sp_SetMark
@name nvarchar (128)
AS
BEGIN TRANSACTION @name WITH MARK
UPDATE coyote.dbo.Marks SET one = 1
COMMIT TRANSACTION;
GO
```

Next, create stored procedure `sp_MarkAll` containing a transaction that places a mark in every database. `sp_MarkAll` can be run from any of the instances.

```
CREATE PROCEDURE sp_MarkAll
```

```
@name nvarchar (128)
AS
BEGIN TRANSACTION
EXEC instance0.coyote.dbo.sp_SetMark @name
EXEC instance1.coyote.dbo.sp_SetMark @name
EXEC instance2.coyote.dbo.sp_SetMark @name
COMMIT TRANSACTION;
GO
```

Two-Phase Commit

Committing a distributed transaction occurs in two phases: prepare and commit. When a marked transaction is committed, the commit log record for each database in the marked transaction is placed in the log at a point where there are no in-doubt transactions in any of the logs. At this point, it is guaranteed that there are no transactions that appear as committed in one log, but not committed in another log.

The following steps accomplish this during the commit of a marked transaction:

1. Prepare phase of a marking transaction stalls all new prepares and commits.
2. Only commits of already prepared transactions are allowed to continue.
3. Marking transaction then waits for all prepared transactions to drain (with time-out).
4. Marked transaction is prepared and committed.
5. The stall of new prepares and commits is removed.

The stalls generated by marked transactions that span multiple databases can reduce the transaction processing performance of the server.

We recommend that you do not run concurrent marked transactions. It is rare but possible for the commit of a distributed marked transaction to deadlock with other distributed marked transactions that are committing at the same time. When this happens, the marking transaction is chosen as the deadlock victim and is rolled back. When this error occurs, the application can retry the marked transaction. When multiple marked transactions try to commit concurrently, there is a higher probability of deadlock.

Recovering to a Marked Transaction

For information about how to recover a database that contains marked transactions to or just before a particular mark, see [Recovering to a Marked Transaction](#).

See Also

[BEGIN DISTRIBUTED TRANSACTION \(Transact-SQL\)](#)

[Back Up and Restore of System Databases \(SQL Server\)](#)

[BEGIN TRANSACTION \(Transact-SQL\)](#)

[Applying Transaction Log Backups](#)

[Full Database Backups](#)

Restore a Database to a Marked Transaction (SQL Server Management Studio)

When a database is in the restoring state, you can use the **Restore Transaction Log** dialog box to restore the database to a marked transaction in the available log backups.

 **Note**

For more information, see [Using Marked Transactions \(Full Recovery Model\)](#) and [Recovering to a Marked Transaction](#).

Procedures

▶ **To restore a marked transaction**

1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**, and, depending on the database, either select a user database or expand **System Databases** and select a system database.
3. Right-click the database, point to **Tasks**, and then click **Restore**.
4. Click **Transaction Log**, which opens the **Restore Transaction Log** dialog box.
5. On the **General** page, in the **Restore To** section, select **Marked transaction**, which opens the **Select Marked Transaction** dialog box. This dialog box displays a grid listing the marked transactions available in the selected transaction log backups.

By default, the restore is up to, but excluding, the marked transaction. To restore the marked transaction also, select **Include marked transaction**.

The following table lists the column headers of the grid and describes their values.

Header	Value
<blank>	Displays a checkbox for selecting the mark.
Transaction Mark	Name of the marked transaction specified by the user when the transaction was committed.
Date	Date and time of the transaction when

	it was committed. Transaction date and time are displayed as recorded in the msdb gmarkhistory table, not in the client computer's date and time.
Description	Description of marked transaction specified by the user when the transaction was committed (if any).
LSN	Log sequence number of the marked transaction.
Database	Name of the database where the marked transaction was committed.
User Name	Name of the database user who committed the marked transaction.

See Also

[SQL Server Management Studio Tutorial](#)

[How to: Restore a Transaction Log Backup \(SQL Server Management Studio\)](#)

Recover to a Log Sequence Number

This topic is relevant only for databases that are using the full or bulk-logged recovery models.

You can use a log sequence number (LSN) to define the recovery point for a restore operation. However, this is a specialized feature that is intended for tools vendors and is unlikely to be generally useful.

Overview of Log Sequence Numbers

LSNs are used internally during a RESTORE sequence to track the point in time to which data has been restored. When a backup is restored, the data is restored to the LSN corresponding to the point in time at which the backup was taken. Differential and log backups advance the restored database to a later time, which corresponds to a higher LSN.

Every record in the transaction log is uniquely identified by a log sequence number (LSN). LSNs are ordered such that if LSN2 is greater than LSN1, the change described by the log record referred to by LSN2 occurred after the change described by the log record LSN.

The LSN of a log record at which a significant event occurred can be useful for constructing correct restore sequences. Because LSNs are ordered, they can be compared for equality and inequality (that is, <, >, =, <=, >=). Such comparisons are useful when constructing restore sequences.

 **Note**

LSNs are values of data type **numeric(25,0)**. Arithmetic operations (for example, addition or subtraction) are not meaningful and must not be used with LSNs.

Viewing LSNs Used by Backup and Restore

The LSN of a log record at which a given backup and restore event occurred is viewable using one or more of the following:

- [backupset](#)
- [backupfile](#)
- [sys.database files](#); [sys.master files](#)
- [RESTORE HEADERONLY](#)
- [RESTORE FILELISTONLY](#)

 **Note**

LSNs also appear in some message texts.

Transact-SQL Syntax for Restoring to an LSN

By using a [RESTORE](#) statement, you can stop at or immediately before the LSN, as follows:

- Use the `WITH STOPATMARK = 'lsn:<lsn_number>'` clause, where `lsn:<lsnNumber>` is a string that specifies that the log record that contains the specified LSN is the recovery point.
STOPATMARK roll forwards to the LSN and includes that log record in the roll forward.
- Use the `WITH STOPBEFOREMARK = 'lsn:<lsn_number>'` clause, where `lsn:<lsnNumber>` is a string that specifies that the log record immediately before the log record that contains the specified LSN number is the recovery point.
STOPBEFOREMARK rolls forward to the LSN and excludes that log record from the roll forward.

Typically, a specific transaction is selected to be included or excluded. Although not required, in practice, the specified log record is a transaction-commit record.

Examples

The following example assumes that the `AdventureWorks` database has been changed to use the full recovery model.

```
RESTORE LOG AdventureWorks FROM DISK = 'c:\adventureworks_log.bak'  
WITH STOPATMARK = 'lsn:15000000040000037'  
GO
```

Related Tasks

- [Restore a Database Backup \(SQL Server Management Studio\)](#)
- [Back Up a Transaction Log \(SQL Server Management Studio\)](#)
- [Restore a Transaction Log Backup \(SQL Server\)](#)
- [Restore to the Point of Failure Under the Full Recovery Model \(Transact-SQL\)](#)
- [Restore a Database to a Marked Transaction \(SQL Server\)](#)
- [Restore a SQL Server Database to a Point in Time \(Full Recovery Model\)](#)

See Also

[Applying Transaction Log Backups](#)

[Transaction Logs \(SQL Server\)](#)

[RESTORE \(Transact-SQL\)](#)

Online Restore

Online restore is supported only on SQL Server 2005 Enterprise Edition and later versions. In this edition, a file, page, or piecemeal restore is online by default. This topic is relevant for databases that contain multiple files or filegroups (and, under the simple recovery model, only for read-only filegroups).

Restoring data while the database is online is called an *online restore*. A database is considered to be online whenever the primary filegroup is online, even if one or more of its secondary filegroups are offline. Under any recovery model, you can restore a file that is offline while the database is online. Under the full recovery model, you can also restore pages while the database is online.



Note

Online restore occurs automatically on SQL Server 2005 Enterprise Edition and later versions and requires no user action. If you do not want to use online restore, you can take a database offline before you start a restore. For more information, see [Taking a Database or File Offline](#), later in this topic.

During an online file restore, any file being restored and its filegroup are offline. If any of these files is online when an online restore starts, the first restore statement takes the filegroup of the file offline. In contrast, during an online page restore, only the page is offline.

Every online restore scenario involves the following basic steps:

1. Restore the data.
2. Restore the log by using `WITH RECOVERY` for the last log restore. This brings the restored data online.

Occasionally, an uncommitted transaction cannot be rolled back because the data that is required by rollback is offline during startup. In this case, the transaction is deferred. For more information, see [Deferred Transactions](#).

 **Note**

If the database is currently using the bulk-logged recovery model, we recommend that you switch to the full recovery model before you start an online restore. For more information, see [View or Change the Recovery Model of a Database \(SQL Server\)](#).

 **Important**

If the backups were taken with multiple devices that were attached to the server, the same number of devices must be available during an online restore.

Log Backups for Online Restore

In an online restore, the recovery point is the point when the data being restored was taken offline or made read-only for the last time. The transaction log backups leading up to and including this recovery point must all be available. Generally, a log backup is required after that point to cover the recovery point for the file. The only exception is during an online restore of read-only data from a data backup that was taken after the data became read-only. In this case, you do not have to have a log backup.

Generally, you may take transaction log backups while the database is online, even after the start of the restore sequence. The timing of the last log backup depends on the properties of the file being restored:

- For an online read-only file, you can take the last log backup that is required for recovery before or during the first restore sequence. A read-only filegroup may not require log backups if a data or differential backup was taken after the filegroup became read-only.

 **Note**

The preceding information also applies to every offline file.

- A special case exists for a read/write file that was online when the first restore statement was issued and that was then automatically taken offline by that restore statement. In this case, you must take a log backup during the first *restore sequence* (the sequence of one or more RESTORE statements that restore, roll forward, and recover data). Generally, this log backup must occur after you restore all the full backups and before you recover the data. However, if there are multiple file backups for a specific filegroup, the minimal point of log backup is the time after the filegroup is offline. This post-data-restore log backup captures the point at which the file was taken offline. The post-data-restore log backup is necessary because the SQL Server Database Engine cannot use online log for an online restore.

 **Note**

Alternatively, you can manually take the file offline before the restore sequence. For more information, see "Taking a Database or File Offline" later in this topic.



Taking a Database or File Offline

If you do not want to use online restore, you can take the database offline before you start the restore sequence by using one of the following methods:

- Under any recovery model, you can take the database offline by using the following [ALTER DATABASE](#) statement:
`ALTER DATABASE database_name SET OFFLINE`
- Alternatively, under the full recovery model, you can force a file or page restore to be offline, by using the following [BACKUP LOG](#) statement put the database in to the restoring state:
`BACKUP LOG database_name WITH NORECOVERY.`

As long as a database remains offline, all restores are offline restores.



Examples



Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)



Related Tasks

- [Restore Files and Filegroups \(SQL Server\)](#)
- [Restore Pages \(SQL Server\)](#)
- [Manage the suspect_pages Table \(SQL Server\)](#)
- [Recover a Database Without Restoring Data \(Transact-SQL\)](#)
- [Remove Defunct Filegroups \(SQL Server\)](#)



See Also

[Restoring File Backups \(Full Recovery Model\)](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

[Performing Page Restores](#)

[Performing Piecemeal Restores](#)

[Restore and Recovery Overview \(SQL Server\)](#)

Deferred Transactions

In SQL Server 2005 Enterprise Edition and later versions, a corrupted transaction can become deferred if data required by rollback (undo) is offline during database startup. A *deferred transaction* is a transaction that is uncommitted when the roll forward phase finishes and that has encountered an error that prevents it from being rolled back. Because the transaction cannot be rolled back, it is deferred.



Note

Corrupted transactions are deferred only in SQL Server 2005 Enterprise Edition and later versions. In other editions of SQL Server, a corrupted transaction causes startup to fail.

Generally, a deferred transaction occurs because, while the database was being rolled forward, an I/O error prevented reading a page that was required by the transaction. However, an error at the file level can also cause deferred transactions. A deferred transaction can also occur when a partial restore sequence stops at a point at which transaction rollback is necessary and a transaction requires data that is offline.

User transactions that are rolling back and hit an I/O error cause the whole database to go offline. When the database is brought back online, the redo reacquires all the locks it had and tries to roll back all the uncommitted transactions. All data modified by a transaction remains appropriately locked until the transaction can roll back. Transactions that cannot be rolled back will give up their locks when the corruption is fixed and the database restarted or, after an online restore, when the deferred transactions are resolved while the database remains online. Until that point, a deferred transaction can hold locks that prevent certain operations on the database as a whole. For example, if a deferred transaction contains a CREATE TABLE instruction, no user can create a table until the deferred transaction has been resolved.

Deferred transaction can also occur because a piecemeal restore recovers a database to a point at which one or more active transactions are affecting a filegroup that has not yet been restored and is offline. Because the transactions cannot be rolled back, they become deferred.

The following table lists the actions that cause a database to perform recovery and the outcome if an I/O problem occurs.

Action	Resolution (if I/O problems occur or required data is offline)
Server start	Deferred transaction
Restore	Deferred transaction
Attach	Attach fails
Autorestart	Deferred transaction
Create database or database snapshot	Creation fails
Redo on database mirroring	Deferred transaction
Filegroup is offline	Deferred transaction

Moving a Transaction Out of the DEFERRED State

Important

Deferred transactions keep the transaction log active. A virtual log file that contains any deferred transactions cannot be truncated until those transactions are moved out of the deferred state. For more information about log truncation, see [Transaction Log \(SQL Server\)](#).

To move the transaction out of the deferred state, the database must start cleanly without any I/O errors. If deferred transactions exist, you must fix the source of the I/O errors. The available solutions, listed in the order in which they are typically tried, are as follows:

- Restart the database. If the problem was transient, the database should start without deferred transactions.
- If the transactions were deferred because a filegroup was offline, bring the filegroup back online.

To bring an offline filegroup back online, use the following Transact-SQL statement:

```
RESTORE DATABASE database_name FILEGROUP=<filegroup_name>
```

- Restore the database. After an online restore, any deferred transactions are resolved. Under the full or bulk-logged recovery model, if the deferred transactions were caused by only a few corrupted pages, an online page restore might resolve the errors (where supported).
- If you are no longer require a filegroup whose offline status is causing deferred transactions, make the offline filegroup defunct. Transactions that were deferred

because the filegroup was offline are moved out of the deferred state after the filegroup becomes defunct.

Important

A defunct filegroup can never be recovered.

For more information, see [Defunct Filegroups](#).

- If transactions were deferred because of a bad page and if a good backup of the database does not exist, use the following process to repair the database:
 - First put the database into emergency mode by executing the following Transact-SQL statement:

```
ALTER DATABASE <database_name> SET EMERGENCY
```

For information about emergency mode, see [Database States](#).

- Then, repair the database by using the DBCC REPAIR_ALLOW_DATA_LOSS option in one of the following DBCC statements: [DBCC CHECKDB](#), [DBCC CHECKALLOC](#), or [DBCC CHECKTABLE](#).

When DBCC encounters the bad page, DBCC deallocates it and repairs any related errors. This approach enables the database to be brought back online in a physically consistent state. However, additional data might also be lost; therefore, this approach should be used as a last resort.

See Also

[Restore and Recovery Overview \(SQL Server\)](#)

[Defunct Filegroups](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

[Performing Page Restores](#)

[Performing Piecemeal Restores](#)

[ALTER DATABASE \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

Remove Defunct Filegroups

This topic describes how to remove defunct filegroups in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL.

In This Topic

- **Before you begin:**
 - Limitations and Restrictions
- Recommendations
 - Security

- **To remove defunct filegroups, using:**

SQL Server Management Studio

Transact-SQL

Before You Begin

Limitations and Restrictions

- This topic is relevant for SQL Server databases that contain multiple files or filegroups; and, under the simple model, only for read-only filegroups.
- All files in a filegroup become defunct when an offline filegroup is removed.

Recommendations

- If an unrestored filegroup will never have to be restored, you can make the filegroup *defunct* by removing it from the database. The defunct filegroup can never be restored to this database, but its metadata remains. After the filegroup is defunct, the database can be restarted, and recovery will make the database consistent across the restored filegroups.

For example, making a filegroup defunct is an option for resolving deferred transactions that were caused by an offline filegroup that you no longer want in the database. Transactions that were deferred because the filegroup was offline are moved out of the deferred state after the filegroup becomes defunct. For more information, see [Deferred Transactions](#).

Security

Permissions

Requires ALTER permission on the database.



Using SQL Server Management Studio

▶ To remove defunct filegroups

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine and then expand that instance.
2. Expand **Databases**, right-click the database from which to delete the file, and then click **Properties**.
3. Select the **Files** page.
4. In the **Database files** grid, select the files to delete, click **Remove**, and then click

OK.

5. Select the **Filegroups** page.
6. In the **Rows** grid, select the filegroup to delete, click **Remove**, and then click **OK**.



Using Transact-SQL

▶ To remove defunct filegroups

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. (**Note:** This example assumes that the files and filegroup already exist. To create these objects, see example B in the [ALTER DATABASE File and Filegroup Options](#) topic.) The first example removes the `test1dat3` and `test1dat4` files from the defunct filegroup by using the `ALTER DATABASE` statement with the `REMOVE FILE` clause. The second example removes the defunct filegroup `Test1FG1` by using the `REMOVE FILEGROUP` clause.

```
USE master;
```

```
GO
```

```
ALTER DATABASE AdventureWorks2012
```

```
REMOVE FILE test1dat3 ;
```

```
ALTER DATABASE AdventureWorks2012
```

```
REMOVE FILE test1dat4 ;
```

```
GO
```

```
USE master;
```

```
GO
```

```
ALTER DATABASE AdventureWorks2012
```

```
REMOVE FILEGROUP Test1FG1 ;
```

```
GO
```



See Also

[ALTER DATABASE File and Filegroup Options \(Transact-SQL\)](#)

[Deferred Transactions](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

[Performing Online Restores](#)

[Performing Page Restores](#)

[Performing Piecemeal Restores](#)

Example: Online Restore of a Read/Write File (Full Recovery Model)

This topic is relevant for SQL Server databases under the full recovery model that contain multiple files or filegroups.

In this example, a database named `adb`, which uses the full recovery model, contains three filegroups. Filegroup `A` is read/write, and filegroup `B` and filegroup `C` are read-only. Initially, all of the filegroups are online.

File `a1` in filegroup `A` appears to be damaged, and the database administrator decides to restore it while the database remains online.



Note

Under the simple recovery model, online restore of read/write data is not allowed.

Restore Sequences



Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

1. Online restore of file `a1`.

```
RESTORE DATABASE adb FILE='a1' FROM backup
WITH NORECOVERY;
```

At this point, file `a1` is in the `RESTORING` state, and filegroup `A` is offline.

2. After restoring the file, the database administrator takes a new log backup to make sure that the point at which the file went offline is captured.

```
BACKUP LOG adb TO log_backup3;
```

3. Online restore of log backups.

The administrator restores all the log backups taken since the restored file backup, ending with the latest log backup (`log_backup3`, taken in step 2). After the last backup is restored, the database is recovered.

```
RESTORE LOG adb FROM log_backup1 WITH NORECOVERY;
RESTORE LOG adb FROM log_backup2 WITH NORECOVERY;
RESTORE LOG adb FROM log_backup3 WITH NORECOVERY;
```

```
RESTORE LOG adb WITH RECOVERY;
```

File a1 is now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Online Restore \(SQL Server\)](#)

[Performing Piecemeal Restores](#)

[BACKUP \(Transact-SQL\)](#)

[Overview of Restore and Recovery in SQL Server](#)

[Applying Transaction Log Backups](#)

[RESTORE \(Transact-SQL\)](#)

Example: Online Restore of a Read-Only File (Full Recovery Model)

This topic is relevant for SQL Server databases under the full recovery model that contain multiple files or filegroups.

In this example, a database named `adb`, which uses the full recovery model, contains three filegroups. Filegroup `A` is read/write, and filegroup `B` and filegroup `C` are read-only. Initially, all of the filegroups are online.

A read-only file, `b1`, in filegroup `B` of database `adb` has to be restored. A backup was taken since the file became read-only; therefore, log backups are not required. Filegroup `B` is offline for the duration of the restore, but the remainder of the database remains online.

Restore Sequence



Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

To restore the file, the database administrator uses the following restore sequence:

```
RESTORE DATABASE adb FILE='b1' FROM filegroup_B_backup
```

WITH RECOVERY

Filegroup B is now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)

See Also

[Online Restore \(SQL Server\)](#)

[Overview of Restore and Recovery in SQL Server](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[RESTORE \(Transact-SQL\)](#)

Example: Online Restore of a Read-Only File (Simple Recovery Model)

This topic is relevant for SQL Server databases under the simple recovery model that contain a read-only filegroup. Under the simple recovery model, a read-only file can be restored online if a file backup exists that was taken since the file became read-only for the last time.

In this example, a database named `adb` contains three filegroups. Filegroup `A` is read/write, and filegroups `B` and `C` are read-only. Initially, all of the filegroups are online. A read-only file in filegroup `B`, `b1`, has to be restored. The database administrator can restore it by using a backup that was taken after the file became read-only. For the duration of the restore, filegroup `B` will be offline, but the remainder of the database will remain online.

Restore Sequence

Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

To restore the file, the database administrator uses the following restore sequence:

```
RESTORE DATABASE adb FILE='b1' FROM filegroup_B_backup
```

WITH RECOVERY

The file is now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Online Restore \(SQL Server\)](#)

[Performing Piecemeal Restores](#)

[Restoring File Backups \(Simple Recovery Model\)](#)

[Overview of Restore and Recovery in SQL Server](#)

[RESTORE \(Transact-SQL\)](#)

Example: Offline Restore of Primary and One Other Filegroup (Full Recovery Model)

This topic is relevant only for databases under the full recovery model that contain multiple filegroups.

In this example, a database named `adb` contains three filegroups. Filegroups `A` and `C` are read/write, and filegroup `B` is read-only. The primary filegroup and filegroup `B` are damaged, but filegroups `A` and `C` are intact. Before the disaster, all the filegroups were online.

The database administrator decides to restore and recover the primary filegroup and filegroup `B`. The database is using the full recovery model; therefore, before the restore starts, a tail-log backup must be taken of the database. When the database comes on line, Filegroups `A` and `C` are automatically brought online.



Note

The offline restore sequence has fewer steps than an online restore of a read-only file. For an example, see [RESTORE \(Transact-SQL\)](#). However, the whole database is offline for the duration of the sequence.

Tail-Log Backup

Before restoring the database, the database administrator must back up the tail of the log. Because the database is damaged, creating the tail-log backup requires using the `NO_TRUNCATE` option:

```
BACKUP LOG adb TO tailLogBackup
```

```
WITH NORECOVERY, NO_TRUNCATE
```

The tail-log backup is the last backup that is applied in the following restore sequences.

Restore Sequence

To restore the primary filegroup and filegroup B, the database administrator uses a restore sequence without the PARTIAL option, as follows:

```
RESTORE DATABASE adb FILEGROUP='Primary' FROM backup1
```

```
WITH NORECOVERY
```

```
RESTORE DATABASE adb FILEGROUP='B' FROM backup2
```

```
WITH NORECOVERY
```

```
RESTORE LOG adb FROM backup3 WITH NORECOVERY
```

```
RESTORE LOG adb FROM backup4 WITH NORECOVERY
```

```
RESTORE LOG adb FROM backup5 WITH NORECOVERY
```

```
RESTORE LOG adb FROM tailLogBackup WITH RECOVERY
```

The files that are not restored are automatically brought online. All the filegroups are now online.

See Also

[Performing Online Restores](#)

[Performing Piecemeal Restores](#)

[Restoring File Backups \(Full Recovery Model\)](#)

[Applying Transaction Log Backups](#)

[RESTORE \(Transact-SQL\)](#)

Restore Pages

This topic describes how to restore pages in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. The goal of a page restore is to restore one or more damaged pages without restoring the whole database. Typically, pages that are candidates for restore have been marked as "suspect" because of an error that is encountered when accessing the page. Suspect pages are identified in the [suspect_pages](#) table in the **msdb** database.

In This Topic

- **Before you begin:**

 - When is a Page Restore Useful?

 - Limitations and Restrictions

 - Recommendations

Security

- **To restore pages, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

When is a Page Restore Useful?

A page restore is intended for repairing isolated damaged pages. Restoring and recovering a few individual pages might be faster than a file restore, reducing the amount of data that is offline during a restore operation. However, if you have to restore more than a few pages in a file, it is generally more efficient to restore the whole file. For example, if lots of pages on a device indicate a pending device failure, consider restoring the file, possibly to another location, and repairing the device.

Furthermore, not all page errors require a restore. A problem can occur in cached data, such as a secondary index, that can be resolved by recalculating the data. For example, if the database administrator drops a secondary index and rebuilds it, the corrupted data, although fixed, is not indicated as such in the [suspect_pages](#) table.

Limitations and Restrictions

- Page restore applies to SQL Server databases that are using the full or bulk-logged recovery models. Page restore is supported only for read/write filegroups.
- Only database pages can be restored. Page restore cannot be used to restore the following:
 - Transaction log
 - Allocation pages: Global Allocation Map (GAM) pages, Shared Global Allocation Map (SGAM) pages, and Page Free Space (PFS) pages.
 - Page 0 of all data files (the file boot page)
 - Page 1:9 (the database boot page)
 - Full-text catalog
- Only database pages can be restored. Page restore cannot be used to restore the following:
 - Transaction log
 - Allocation pages: Global Allocation Map (GAM) pages, Shared Global Allocation Map (SGAM) pages, and Page Free Space (PFS) pages.
 - Page 0 of all data files (the file boot page)
 - Page 1:9 (the database boot page)

- Full-text catalog
- For a database that uses the bulk-logged recovery model, page restore has the following additional conditions:
 - Backing up while filegroup or page data is offline is problematic for bulk-logged data, because the offline data is not recorded in the log. Any offline page can prevent backing up the log. In this cases, consider using DBCC REPAIR, because this might cause less data loss than restoring to the most recent backup.
 - If a log backup of a bulk-logged database encounters a bad page, it fails unless WITH CONTINUE_AFTER_ERROR is specified.
 - Page restore generally does not work with bulk-logged recovery.
A best practice for performing page restore is to set the database to the full recovery model, and try a log backup. If the log backup works, you can continue with the page restore. If the log backup fails, you either have to lose work since the previous log backup or you have to try running DBCC must be run with the REPAIR_ALLOW_DATA_LOSS option.

Recommendations

- Page restore scenarios:

Offline page restore

All editions of SQL Server 2005 and later versions support restoring pages when the database is offline. In an offline page restore, the database is offline while damaged pages are restored. At the end of the restore sequence, the database comes online.

Online page restore

SQL Server 2005 Enterprise Edition and later versions support online page restores, though they use offline restore if the database is currently offline. In most cases, a damaged page can be restored while the database remains online, including the filegroup to which a page is being restored. When the primary filegroup is online, even if one or more of its secondary filegroups are offline, page restores are usually performed online. Occasionally, however, a damaged page can require an offline restore. For example, damage to certain critical pages might prevent the database from starting.

Warning

If damaged pages are storing critical database metadata, required updates to metadata might fail during an online page restore attempt. In this case, you can perform an offline page restore, but first you must create a [tail log backup](#) (by backing up the transaction log using RESTORE WITH

NORECOVERY).

- Page restore takes advantage of the improved page-level error reporting (including page checksums) and tracking that was introduced in SQL Server 2005. Pages that are detected as corrupted by check-summing or a torn write, *damaged pages*, can be restored by a page restore operation. Only explicitly specified pages are restored. Each specified page is replaced by the copy of that page from the specified data backup.

When you restore the subsequent log backups, they are applied only to database files that contain at least one page that is being recovered. An unbroken chain of log backups must be applied to the last full or differential restore to bring the filegroup that contains the page forward to the current log file. As in a file restore, the roll forward set is advanced with a single log redo pass. For a page restore to succeed, the restored pages must be recovered to a state consistent with the database.

Security

Permissions

If the database being restored does not exist, the user must have CREATE DATABASE permissions to be able to execute RESTORE. If the database exists, RESTORE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles and the owner (**dbo**) of the database (for the FROM DATABASE_SNAPSHOT option, the database always exists).

RESTORE permissions are given to roles in which membership information is always readily available to the server. Because fixed database role membership can be checked only when the database is accessible and undamaged, which is not always the case when RESTORE is executed, members of the **db_owner** fixed database role do not have RESTORE permissions.



Using SQL Server Management Studio

Starting in SQL Server 2012, SQL Server Management Studio supports page restores.

▶ To restore pages

1. Connect to the appropriate instance of the SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.
2. Expand **Databases**. Depending on the database, either select a user database or expand **System Databases**, and then select a system database.
3. Right-click the database, point to **Tasks**, point to **Restore**, and then click **Page**,

which opens the **Restore Page** dialog box.

Restore

This section performs the same function as that of **Restore to** on the [Restore Database \(General Page\)](#).

Database

Specifies the database to restore. You can enter a new database or select an existing database from the drop-down list. The list includes all databases on the server, except the system databases **master** and **tempdb**.

Warning

To restore a password-protected backup, you must use the [RESTORE](#) statement.

Tail-Log backup

Enter or select a file name in **Backup device** where there tail-log backup will be stored for the database.

Backup Sets

This section displays the backup sets involved in the restoration.

Header	Values
Name	The name of the backup set.
Component	The backed-up component: Database, File, or <blank> (for transaction logs).
Type	The type of backup performed: Full, Differential, or Transaction Log .
Server	The name of the Database Engine instance that performed the backup operation.
Database	The name of the database involved in the backup operation.
Position	The position of the backup set in the volume.
First LSN	The log sequence number (LSN) of the first transaction in the backup set. Blank for file backups.

Last LSN	The log sequence number (LSN) of the last transaction in the backup set. Blank for file backups.
Checkpoint LSN	The log sequence number (LSN) of the most recent checkpoint at the time the backup was created.
Full LSN	The log sequence number (LSN) of the most recent full database backup.
Start Date	The date and time when the backup operation began, presented in the regional setting of the client.
Finish Date	The date and time when the backup operation finished, presented in the regional setting of the client.
Size	The size of the backup set in bytes.
User Name	The name of the user who performed the backup operation.
Expiration	The date and time the backup set expires.

Click **Verify** to check the integrity of the backup files needed to perform the page restore operation.

- To identify corrupted pages, with the correct database selected in the **Database** box, click **Check Database Pages**. This is a long running operation.

wWarning

To restore specific pages that are not corrupted, click **Add** and enter the **File ID** and **Page ID** of the pages to be restored.

- The pages grid is used to identify the pages to be restored. Initially, this grid is populated from the [suspect_pages](#) system table. To add or remove pages from the grid, click **Add** or **Remove**. For more information, see [Understanding and Managing the suspect_pages Table](#).
- The **Backup sets** grid lists the backup sets in the default restore plan. Optionally, click **Verify** to verify that the backups are readable and that the backup sets are complete, without restoring them. For more information, see [RESTORE VERIFYONLY \(Transact-SQL\)](#).

Pages

7. To restore the pages listed in the pages grid, click **OK**.



Using Transact-SQL

To specify a page in a RESTORE DATABASE statement, you need the file ID of the file containing the page and the page ID of the page. The required syntax is as follows:

```
RESTORE DATABASE <database_name>  
    PAGE = '<file: page> [ ,... n ] ' [ ,... n ]  
    FROM <backup_device> [ ,... n ]  
WITH NORECOVERY
```

For more information about the parameters of the PAGE option, see [RESTORE Arguments \(Transact-SQL\)](#). For more information about the RESTORE DATABASE syntax, see [RESTORE \(Transact-SQL\)](#).

▶ To restore pages

1. Obtain the page IDs of the damaged pages to be restored. A checksum or torn write error returns page ID, providing the information required for specifying the pages. To look up page ID of a damaged page, use any of the following sources.

Source of page ID	Topic
msdb..suspect_pages	Understanding and Managing the suspect_pages Table
Error log	Viewing the SQL Server Error Log
Event traces	Monitor and Respond to Events
DBCC	DBCC (Transact-SQL)
WMI provider	WMI Provider for Server Events Concepts

2. Start a page restore with a full database, file, or filegroup backup that contains the page. In the RESTORE DATABASE statement, use the PAGE clause to list the page IDs of all of the pages to be restored.
3. Apply the most recent differentials .
4. Apply the subsequent log backups.
5. Create a new log backup of the database that includes the final LSN of the restored pages, that is, the point at which the last restored page is taken offline.

The final LSN, which is set as part of the first restore in the sequence, is the redo target LSN. Online roll forward of the file containing the page is able to stop at the redo target LSN. To learn the current redo target LSN of a file, see the **redo_target_lsn** column of **sys.master_files**. For more information, see [sys.master_files \(Transact-SQL\)](#).

6. Restore the new log backup. After this new log backup is applied, the page restore is completed and the pages are now usable.

nNote

This sequence is analogous to a file restore sequence. In fact, page restore and file restores can both be performed as part of the same sequence.

Example (Transact-SQL)

The following example restores four damaged pages of file B with `NORECOVERY`. Next, two log backups are applied with `NORECOVERY`, followed with the tail-log backup, which is restored with `RECOVERY`. This example performs an online restore. In the example, the file ID of file B is 1, and the page IDs of the damaged pages are 57, 202, 916, and 1016.

```
RESTORE DATABASE <database> PAGE='1:57, 1:202, 1:916, 1:1016'  
    FROM <file_backup_of_file_B>  
    WITH NORECOVERY;  
RESTORE LOG <database> FROM <log_backup>  
    WITH NORECOVERY;  
RESTORE LOG <database> FROM <log_backup>  
    WITH NORECOVERY;  
BACKUP LOG <database> TO <new_log_backup>;  
RESTORE LOG <database> FROM <new_log_backup> WITH RECOVERY;  
GO
```



See Also

[RESTORE \(Transact-SQL\)](#)

[Applying Transaction Log Backups](#)

[The suspect pages](#)

[Back Up and Restore of SQL Server Databases](#)

Manage the suspect_pages Table

This topic describes how to manage the **suspect_pages** table in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. The **suspect_pages** table is used for maintaining information about suspect pages, and is relevant in helping to decide whether a restore is necessary. The [suspect_pages](#) table resides in the [msdb database](#) and was introduced in SQL Server 2005.

A page is considered "suspect" when the SQL Server Database Engine encounters one of the following errors when it tries to read a data page:

- An [823 error](#) that was caused by a cyclic redundancy check (CRC) issued by the operating system, such as a disk error (certain hardware errors)
- An [824 error](#), such as a torn page (any logical error)

The page ID of every suspect page is recorded in the **suspect_pages** table. The Database Engine records any suspect pages encountered during regular processing, such as the following:

- A query has to read a page.
- During a DBCC CHECKDB operation.
- During a backup operation.

The **suspect_pages** table is also updated as necessary during a restore operation, a DBCC repair operation, or a drop database operation.

In This Topic

- **Before you begin:**
 - Recommendations
 - Security
- **To manage the suspect_pages table, using:**
 - SQL Server Management Studio
 - Transact-SQL

Before You Begin

Recommendations

- **Errors Recorded in suspect_pages Table**

The **suspect_pages** table contains one row per page that failed with an 824 error, up to a limit of 1,000 rows. The following table shows errors logged in the **event_type** column of the **suspect_pages** table.

Error description	event_type value
823 error caused by an operating system CRC error or 824 error other than a bad checksum or a torn page (for example, a bad page ID)	1
Bad checksum	2
Torn page	3
Restored (The page was restored after it was marked bad)	4
Repaired (DBCC repaired the page)	5
Deallocated by DBCC	7

The **suspect_pages** table also records transient errors. Sources of transient errors include an I/O error (for example, a cable was disconnected) or a page that temporarily fails a repeated checksum test.

- **How the Database Engine Updates the suspect_pages Table**

The Database Engine takes the following actions on the **suspect_pages** table:

- If the table is not full, it is updated for every 824 error, to indicate that an error has occurred, and the error counter is incremented. If a page has an error after it is fixed by being repaired, restored, or deallocated, its **number_of_errors** count is incremented and its **last_update** column is updated
- After a listed page is fixed by a restore or a repair operation, the operation updates the **suspect_pages** row to indicate that the page is repaired (**event_type** = 5) or restored (**event_type** = 4).
- If a DBCC check is run, the check marks any error-free pages as repaired (**event_type** = 5) or deallocated (**event_type** = 7).

- **Automatic Updates to the suspect_pages Table**

A database mirroring partner or AlwaysOn availability replica updates the **suspect_pages** table after an attempt to read a page from a data file fails for one of the following reasons.

- An 823 error that is caused by an operating system CRC error.
- An 824 error (logical corruption such as a torn page).

The following actions also automatically update rows in the **suspect_pages** table.

- DBCC CHECKDB REPAIR_ALLOW_DATA_LOSS updates the **suspect_pages** table to indicate each page that it has deallocated or repaired.
- A full, file, or page RESTORE marks the page entries as restored.

The following actions automatically delete rows from the **suspect_pages** table.

- ALTER DATABASE REMOVE FILE
 - DROP DATABASE
- **Maintenance Role of the Database Administrator**
Database administrators are responsible for managing the table, primarily by deleting old rows. The **suspect_pages** table is limited in size, and if it fills, new errors are not logged. To prevent this table from filling up, the database administrator or system administrator must manually clear out old entries from this table by deleting rows. Therefore, we recommend that you periodically delete or archive rows that have an **event_type** of restored or repaired, or rows that have an old **last_update** value.

To monitor the activity on the suspect_pages table, you can use the [Database Suspect Data Page Event Class](#). Rows are sometimes added to the **suspect_pages** table because of transient errors. If many rows are being added to the table, however, a problem probably exists with the I/O subsystem. If you notice a sudden increase in the number of rows being added to the table, we recommend that you investigate possible problems in your I/O subsystem.

A database administrator can also insert or update records. For example, updating a row might be useful when the database administrator knows that a particular suspect page is actually intact, but wants to preserve the record for a while.

Security

Permissions

Anyone with access to **msdb** can read the data in the **suspect_pages** table. Anyone with UPDATE permission on the suspect_pages table can update its records. Members the **db_owner** fixed database role on **msdb** or the **sysadmin** fixed server role can insert, update, and delete records.



Using SQL Server Management Studio

▶ To manage the suspect_pages table

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, expand that instance, and then expand **Databases**.
2. Expand **System Databases**, expand **msdb**, expand **Tables**, and then expand

System Tables.

3. Expand **dbo.suspect_pages** and right-click **Edit Top 200 Rows**.
4. In the query window, edit, update, or delete the rows that you want.



Using Transact-SQL

▶ To manage the suspect_pages table

1. Connect to the Database Engine.
2. From the Standard bar, click **New Query**.
3. Copy and paste the following examples into the query window and click **Execute**.
This example deletes some of the rows from the `suspect_pages` table.

```
-- Delete restored, repaired, or deallocated pages.
```

```
DELETE FROM msdb..suspect_pages
    WHERE (event_type = 4 OR event_type = 5 OR event_type = 7);
GO
```

This example returns the bad pages in the `suspect_pages` table.

```
-- Select nonspecific 824, bad checksum, and torn page errors.
```

```
SELECT * FROM msdb..suspect_pages
    WHERE (event_type = 1 OR event_type = 2 OR event_type = 3);
GO
```



See Also

[DROP DATABASE \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[BACKUP \(Transact-SQL\)](#)

[DBCC \(Transact-SQL\)](#)

[Performing Page Restores](#)

[suspect_pages \(Transact-SQL\)](#)

[MSSQLSERVER 823](#)

[MSSQLSERVER 824](#)

Piecemeal Restores

This topic is relevant only for databases in SQL Server 2005 Enterprise Edition and later versions that contain multiple files or filegroups; and, under the simple model, only for read-only filegroups.

Piecemeal restore, which was introduced in SQL Server 2005, allows databases that contain multiple filegroups to be restored and recovered in stages. Piecemeal restore involves a series of restore sequences, starting with the primary filegroup and, in some cases, one or more secondary filegroups. Piecemeal restore maintains checks to ensure that the database will be consistent in the end. After the restore sequence is completed, recovered files, if they are valid and consistent with the database, can be brought online directly.

Piecemeal restore works with all recovery models, but is more flexible for the full and bulk-logged models than for the simple model.

Every piecemeal restore starts with an initial restore sequence called the *partial-restore sequence*. Minimally, the partial-restore sequence restores and recovers the primary filegroup and, under the simple recovery model, all read/write filegroups. During the piecemeal-restore sequence, the whole database must go offline. Thereafter, the database is online and restored filegroups are available. However, any unrestored filegroups remain offline and are not accessible. Any offline filegroups, however, can be restored and brought online later by a file restore.

Regardless of the recovery model that is used by the database, the partial-restore sequence starts with a RESTORE DATABASE statement that restores a full backup and specifies the PARTIAL option. The PARTIAL option always starts a new piecemeal restore; therefore, you must specify PARTIAL only one time in the initial statement of the partial-restore sequence. When the partial restore sequence finishes and the database is brought online, the state of the remaining files becomes "recovery pending" because their recovery has been postponed.

Subsequently, a piecemeal restore typically includes one or more restore sequences, which are called *filegroup-restore sequences*. You can wait to perform a specific filegroup-restore sequence for as long as you want. Each filegroup-restore sequence restores and recovers one or more offline filegroups to a point consistent with the database. The timing and number of filegroup-restore sequences depends on your recovery goal, the number of offline filegroups you want to restore, and on how many of them you restore per filegroup-restore sequence.

The exact requirements for performing a piecemeal restore depend on the recovery model of the database. For more information, see "Piecemeal Restore Under the Simple Recovery Model" and "Piecemeal Restore Under the Full Recovery Model," later in this topic.

Piecemeal Restore Scenarios

All editions of SQL Server support offline piecemeal restores. In SQL Server 2005 Enterprise Edition and later versions, a piecemeal restore can be either online or offline. The implications of offline and online piecemeal restores are as follows:

- Offline piecemeal restore scenario

In an offline piecemeal restore, the database is online after the partial-restore sequence. Filegroups that have not yet been restored remain offline, but they can be restored as you need them after taking the database offline.

- Online piecemeal restore scenario

In an online piecemeal restore, after the partial-restore sequence, the database is online, and the primary filegroup and any recovered secondary filegroups are available. Filegroups that have not yet been restored remain offline, but they can be restored as needed while the database remains online.

Online piecemeal restores can involve deferred transactions. When only a subset of filegroups has been restored, transactions in the database that depend on online filegroups might become deferred. This is typical, because the whole database must be consistent. For more information, see [Deferred Transactions](#).

Restrictions

If a partial restore sequence excludes any [FILESTREAM](#) filegroup, point-in-time restore is not supported. You can force the restore sequence to continue. However the FILESTREAM filegroups that are omitted from your RESTORE statement can never be restored. To force a point-in-time restore, specify the CONTINUE_AFTER_ERROR option together with the STOPAT, STOPATMARK, or STOPBEFOREMARK option, which you must also specify in your subsequent RESTORE LOG statements. If you specify CONTINUE_AFTER_ERROR, the partial restore sequence succeeds and the FILESTREAM filegroup becomes unrecoverable.

Piecemeal Restore Under the Simple Recovery Model

Under the simple recovery model, the piecemeal restore sequence must start with a full database or partial backup. Then, if the restored backup is a differential base, restore the latest differential backup next.

During the first partial restore sequence, if you restore only a subset of read/write filegroups, any unrestored filegroups become defunct when you recover the partially restored database. Omitting a read/write filegroup from the partial-restore sequence is appropriate only in the following cases:

- You intend for the unrestored filegroups to become defunct.
- The restore sequence will arrive at a recovery point at which each unrestored filegroup has become read-only, dropped, or defunct (during a previous restore in the partial-restore sequence).

- The full backup was taken while the database was using the simple recovery model, but the recovery point is at a time when the database is using the full recovery model. For more information, see "Performing a Piecemeal Restore of a Database Whose Recovery Model Has Been Switched from Simple to Full," later in this topic.

Requirements for Piecemeal Restore Under the Simple Recovery Model

Under the simple recovery model, the initial stage restores and recovers the primary filegroup and all read/write secondary filegroups. After the initial stage is completed, recovered files, if they are valid and consistent with the database, can be brought online directly.

Thereafter, read-only filegroups can be restored in one or more additional stages.

Piecemeal restore is available for a read-only secondary filegroup only if the following are true:

- Was read-only when backed up.
- Has remained read-only (keeping it logically consistent with the primary filegroup).

To perform a piecemeal restore, the following guidelines must be followed:

- A complete set of backups for the piecemeal restore of a simple recovery model database must contain the following:
 - A partial or full database backup that contains the primary filegroup and all filegroups that were read/write at the time of the backup.
 - A backup of each read-only file.
- For the backup of a read-only file to be consistent with the primary filegroup, the secondary filegroup must have been read-only from when it was backed up until the backup that contains the primary filegroup was completed. You can use differential file backups, if they were taken after the filegroup became read-only.

Piecemeal Restore Stages (Simple Recovery Model)

The piecemeal restore scenario involves the following stages:

- Initial stage (restore and recover the primary filegroup and all read/write filegroups)

The initial stage performs a partial restore. The partial restore sequence restores the primary filegroup, all read/write secondary filegroups, and (optionally) some of the read-only filegroups. During the initial stage, the whole database must go offline.

After the initial stage, the database is online, and restored filegroups are available. However, any read-only filegroups that have not yet been restored, remain offline.

The first RESTORE statement in the initial stage must do the following:

- Use a partial or full database backup that contains the primary filegroup and all filegroups that were read/write at the time of the backup. It is common to start a partial restore sequence by restoring a partial backup.
- Specify the PARTIAL option, which indicates the start of a piecemeal restore.



Note

In SQL Server 2005 and later versions, the PARTIAL option performs safety checks that ensure that the resulting database is suited for use as a production database.

- Specify the READ_WRITE_FILEGROUPS option if the backup is a full database backup.
- While the database is online, you can use one or more online file restores to restore and recover offline read-only files that were read-only at the time of backup. The timing of the online file restores depends on when you want to have the data online. Whether you must restore data to a file depends on the following:

- Valid read-only files that are consistent with the database can be brought online directly by recovering them without restoring any data.
- Files that are damaged or inconsistent with the database must be restored before they are recovered.

Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Primary and One Other Filegroup \(Simple Recovery Model\)](#)

Piecemeal Restore Under the Full Recovery Model

Under the full recovery model or bulk-logged recovery model, piecemeal restore is available for any database that contains multiple filegroups and you can restore a database to any point in time. The restore sequences of a piecemeal restore behave as follows:

- Partial-restore sequence
The partial restore sequence restores the primary filegroup and, optionally, some of the secondary filegroups.
The first RESTORE DATABASE statement must do the following:
 - Specify the PARTIAL option. This indicates the start of a piecemeal restore.
 - Use any full database backup that contains the primary filegroup. The common practice is to start a partial restore sequence by restoring a partial backup.
 - To restore to a specific point in time, you must specify the time in the partial restore sequence. Every successive step of the restore sequence must specify the same point in time.
- Filegroup-restore sequences bring additional filegroups online to a point consistent with the database.

In SQL Server 2005 Enterprise Edition and later versions, any offline secondary filegroup can be restored and recovered while the database remains online. If a

specific read-only file is undamaged and consistent with the database, the file does not have to be restored. For more information, see [Recovering a Database Without Restoring Data](#).

Applying Log Backups

If a read-only filegroup has been read-only since before the file backup was created, applying log backups to the filegroup is unnecessary and is skipped by file restore. If the filegroup is read/write, an unbroken chain of log backups must be applied to the last full or differential restore to bring the filegroup forward to the current log file.

Examples

- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Primary and One Other Filegroup \(Full Recovery Model\)](#)

Performing a Piecemeal Restore of a Database Whose Recovery Model Has Been Switched from Simple to Full

You can perform a piecemeal restore of a database that has been switched from the simple recovery model to the full recovery model since the full partial or database backup. For example, consider a database for which you take the following steps:

1. Create a partial backup (backup_1) of a simple-model database.
2. After some time, change the recovery model to full.
3. Create a differential backup.
4. Start taking log backups.

Thereafter, the following sequence is valid:

1. A partial restore that omits some secondary filegroups.
2. A differential restore followed by any other needed restores.
3. Later, a file restore of a read/write secondary filegroup WITH NORECOVERY from the backup_1 partial backup
4. The differential backup followed by any other backups that were restored in the original piecemeal restore sequence to restore the data up to the original recovery point.

See Also

[Applying Transaction Log Backups](#)

[RESTORE \(Transact-SQL\)](#)

[Restoring a Database to a Point in Time](#)

[Restore and Recovery Overview](#)

[Plan and Perform Restore Sequences \(Full Recovery Model\)](#)

Example: Piecemeal Restore of Database (Full Recovery Model)

A piecemeal restore sequence restores and recovers a database in stages at the filegroup level, beginning with the primary and all read-write, secondary filegroups.

In this example, database `adb` is restored to a new computer after a disaster. The database is using the full recovery model; therefore, before the restore starts, a tail-log backup must be taken of the database. Before the disaster, all the filegroups are online. Filegroup `B` is read-only. All of the secondary filegroups must be restored, but they are restored in order of importance: `A` (highest), `C`, and lastly `B`. In this example, there are four log backups, including the tail-log backup.

Tail-Log Backup

Before restoring the database, the database administrator must back up the tail of the log. Because the database is damaged, creating the tail-log backup requires using the `NO_TRUNCATE` option:

```
BACKUP LOG adb TO tailLogBackup WITH NORECOVERY, NO_TRUNCATE
```

The tail-log backup is the last backup that is applied in the following restore sequences.

Restore Sequences

Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

1. Partial restore of the primary and secondary filegroup `A`.

```
RESTORE DATABASE adb FILEGROUP='Primary' FROM backup1
    WITH PARTIAL, NORECOVERY
RESTORE DATABASE adb FILEGROUP='A' FROM backup2
    WITH NORECOVERY
RESTORE LOG adb FROM backup3 WITH NORECOVERY
RESTORE LOG adb FROM backup4 WITH NORECOVERY
RESTORE LOG adb FROM backup5 WITH NORECOVERY
RESTORE LOG adb FROM tailLogBackup WITH RECOVERY
```

2. Online restore of filegroup `C`.

At this point, the primary filegroup and secondary filegroup `A` are online. All the files in filegroups `B` and `C` are recovery pending, and the filegroups are offline.

Messages from the last `RESTORE LOG` statement in step 1 indicate that rollback of transactions that involve filegroup `C` was deferred, because this filegroup is not

available. Regular operations can continue, but locks are held by these transactions and log truncation will not occur until the rollback can complete.

In the second restore sequence, the database administrator restores filegroup c:

```
RESTORE DATABASE adb FILEGROUP='C' FROM backup2a WITH NORECOVERY
RESTORE LOG adb FROM backup3 WITH NORECOVERY
RESTORE LOG adb FROM backup4 WITH NORECOVERY
RESTORE LOG adb FROM backup5 WITH NORECOVERY
RESTORE LOG adb FROM tailLogBackup WITH RECOVERY
```

At this point the primary and filegroups A and C are online. Files in filegroup B remain recovery pending, with the filegroup offline. Deferred transactions have been resolved, and log truncation occurs.

3. Online restore of filegroup B.

In the third restore sequence, the database administrator restores filegroup B. The backup of filegroup B was taken after the filegroup became read-only; therefore, it does not have to be rolled forward during recovery.

```
RESTORE DATABASE adb FILEGROUP='B' FROM backup2b WITH RECOVERY
```

All filegroups are now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Performing Piecemeal Restores](#)

[Online Restore \(SQL Server\)](#)

[Applying Transaction Log Backups](#)

[RESTORE \(Transact-SQL\)](#)

[Performing Piecemeal Restores](#)

Example: Piecemeal Restore of Database (Simple Recovery Model)

A piecemeal restore sequence restores and recovers a database in stages at the filegroup level, starting with the primary and all read/write, secondary filegroups.

In this example, database `adb` is restored to a new computer after a disaster. The database is using the simple recovery model. Before the disaster, all the filegroups are online. Filegroups `A` and `C` are read/write, and filegroup `B` is read-only. Filegroup `B` became read-only before the most recent partial backup, which contains the primary filegroup and the read/write secondary filegroups, `A` and `C`. After filegroup `B` became read-only, a separate file backup of filegroup `B` was taken.

Restore Sequences

1. Partial restore of the primary and filegroups `A` and `C`.

```
RESTORE DATABASE adb FILEGROUP='A', FILEGROUP='C'  
FROM partial_backup  
WITH PARTIAL, RECOVERY;
```

At this point, the primary and filegroups `A` and `C` are online. All files in filegroup `B` are recovery pending, and the filegroup is offline.

2. Online restore of filegroup `B`.

```
RESTORE DATABASE adb FILEGROUP='B' FROM backup  
WITH RECOVERY;
```

All filegroups are now online.

Additional Examples

- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Online Restore \(SQL Server\)](#)

[Performing Piecemeal Restores](#)

[RESTORE \(Transact-SQL\)](#)

[Performing Piecemeal Restores](#)

Example: Piecemeal Restore of Only Some Filegroups (Full Recovery Model)

This topic is relevant for SQL Server databases under the full recovery model that contain multiple files or filegroups.

A piecemeal restore sequence restores and recovers a database in stages at the filegroup level, starting with the primary and all read/write, secondary filegroups.

In this example, a database named `adb`, which uses the full recovery model, contains three filegroups. Filegroup `A` is read/write, and filegroup `B` and filegroup `C` are read-only. Initially, all of the filegroups are online.

The primary and filegroup `B` of database `adb` appear to be damaged. The primary filegroup is fairly small and can be restored quickly. The database administrator decides to restore them by using a piecemeal restore sequence. First, the primary filegroup and the subsequent transaction logs are restored the database is recovered.

The intact filegroups `A` and `C` contain critical data. Therefore, they will be recovered next to bring them online as quickly as possible. Finally, the damaged secondary filegroup, `B`, is restored and recovered.

Restore Sequences:



Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

1. Create a tail log backup of database `adb`. This step is essential to make the intact filegroups `A` and `C` current with the recovery point of the database.

```
BACKUP LOG adb TO tailLogBackup WITH NORECOVERY
```

2. Partial restore of the primary filegroup.

```
RESTORE DATABASE adb FILEGROUP='Primary' FROM backup  
WITH PARTIAL, NORECOVERY
```

```
RESTORE LOG adb FROM backup1 WITH NORECOVERY
```

```
RESTORE LOG adb FROM backup2 WITH NORECOVERY
```

```
RESTORE LOG adb FROM backup3 WITH NORECOVERY
```

```
RESTORE LOG adb FROM tailLogBackup WITH RECOVERY
```

At this point the primary is online. Files in filegroups `A`, `B`, and `C` are recovery pending, and the filegroups are offline.

3. Online restore of filegroups `A` and `C`.

Because their data is undamaged, these filegroups do not have to be restored from a backup, but they do have to be recovered to bring them online.

The database administrator recovers A and C immediately.

```
RESTORE DATABASE adb FILEGROUP='A', FILEGROUP='C' WITH RECOVERY
```

At this point the primary and filegroups A and C are online. Files in filegroup B remain recovery pending, with the filegroup offline.

4. Online restore of filegroup B.

Files in filegroup B are restored any time thereafter.



Note

The backup of filegroup B was taken after the filegroup became read-only; therefore, these files do not have to be rolled forward.

```
RESTORE DATABASE adb FILEGROUP='B' FROM backup WITH RECOVERY
```

All filegroups are now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Performing Piecemeal Restores](#)

[Online Restore \(SQL Server\)](#)

[Applying Transaction Log Backups](#)

[RESTORE \(Transact-SQL\)](#)

[Performing Piecemeal Restores](#)

Example: Piecemeal Restore of Only Some Filegroups (Simple Recovery Model)

This topic is relevant for SQL Server databases under the simple recovery model that contain a read-only filegroup.

A piecemeal restore sequence restores and recovers a database in stages at the filegroup level, beginning with the primary and all read/write, secondary filegroups.

In this example, a database named `adb`, which uses the simple recovery model, contains three filegroups. Filegroup A is read/write, and filegroup B and filegroup C are read-only. Initially, all of the filegroups are online.

The primary and filegroup `B` of database `adb` appear to be damaged; therefore, the database administrator decides to restore them by using a piecemeal restore sequence. Under the simple recovery model, all read/write filegroups must be restored from the same partial backup. Although filegroup `A` is intact, it must be restored with the primary filegroup to make sure that they are consistent (the database will be restored to the point in time defined by the end of the last partial backup). Filegroup `C` is intact, but it must be recovered to bring it online. Filegroup `B`, although damaged, contains less critical data than Filegroup `C`; therefore, `B` will be restored last.

Restore Sequences

Note

The syntax for an online restore sequence is the same as for an offline restore sequence.

1. Partial restore of the primary and filegroup `A` from a partial backup.

```
RESTORE DATABASE adb READ_WRITE_FILEGROUPS FROM partial_backup
WITH PARTIAL, RECOVERY
```

At this point the primary filegroup and filegroup `A` are online. Files in filegroups `B` and `C` are recovery pending, and the filegroups are offline.

2. Online recovery of filegroup `C`.

Filegroup `C` is consistent because the partial backup that was restored above was taken after filegroup `C` became read-only, although the database was taken back in time by the restore. The database administrator recovers the filegroup `C`, without restoring it, to bring it online.

```
RESTORE DATABASE adb FILEGROUP='C' WITH RECOVERY
```

At this point the primary and filegroups `A` and `C` are online. Files in filegroup `B` remain recovery pending, with the filegroup offline.

3. Online restore of filegroup `B`.

Files in filegroup `B` must be restored. The database administrator restores the backup of filegroup `B` taken after filegroup `B` became read-only and before the partial backup.

```
RESTORE DATABASE adb FILEGROUP='B' FROM backup
WITH RECOVERY
```

All filegroups are now online.

Additional Examples

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)

- [Example: Online Restore of a Read-Write File \(Full Recovery Model\)](#)
- [Example: Online Restore of a Read-Only File \(Full Recovery Model\)](#)

See Also

[Online Restore \(SQL Server\)](#)

[BACKUP \(Transact-SQL\)](#)

[RESTORE \(Transact-SQL\)](#)

[Performing Piecemeal Restores](#)

Recover a Database Without Restoring Data (Transact-SQL)

Usually, all of the data in a SQL Server database is restored before the database is recovered. However, a restore operation can recover a database without actually restoring a backup; for example, when recovering a read-only file that is consistent with the database. This is referred to as a *recovery-only restore*. When offline data is already consistent with the database and needs only to be made available, a recovery-only restore operation completes the recovery of the database and bring the data online.

A recovery-only restore can occur for a whole database or for one or more a files or filegroups.

Recovery-Only Database Restore

A recovery-only database restore can be useful in the following situations:

- You did not recover the database when restoring the last backup in a restore sequence, and you now want to recover the database to bring it online.
- The database is in standby mode, and you want to make the database updatable without applying another log backup.

The [RESTORE](#) syntax for a recovery-only database restore is as follows:

```
RESTORE DATABASE database_name WITH RECOVERY
```



Note

The FROM = *<backup_device>* clause is not used for recovery-only restores because no backup is necessary.

Example

The following example recovers the `AdventureWorks2012` sample database in a restore operation without restoring data.

```
-- Restore database using WITH RECOVERY.
```

```
RESTORE DATABASE AdventureWorks2012
```

Recovery-Only File Restore

A recovery-only file restore can be useful in the following situation:

A database is restored piecemeal. After restore of the primary filegroup is complete, one or more of the unrestored files are consistent with the new database state, perhaps because it has been read-only for some time. These files only have to be recovered; data copying is unnecessary.

A recovery-only restore operation brings the data in the offline filegroup online; no data-copy, redo, or undo phase occurs. For information about the phases of restore, see [Restore and Recovery Overview \(SQL Server\)](#).

The [RESTORE](#) syntax for a recovery-only file restore is:

```
RESTORE DATABASE database_name { FILE = logical_file_name | FILEGROUP =
logical_filegroup_name } [ ,...n ] WITH RECOVERY
```

Example

The following example illustrates a recovery-only file restore of the files in a secondary filegroup, `SalesGroup2`, in the `Sales` database. The primary filegroup has already been restored as the initial step of a piecemeal restore, and `SalesGroup2` is consistent with the restored primary filegroup. Recovering this filegroup and bringing it online requires only a single statement.

```
RESTORE DATABASE Sales FILEGROUP=SalesGroup2 WITH RECOVERY;
```

Examples of Completing a Piecemeal Restore Scenario with a Recovery-Only Restore

Simple recovery model

- [Example: Piecemeal Restore of Database \(Simple Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Simple Recovery Model\)](#)

Full recovery model

- [Example: Piecemeal Restore of Database \(Full Recovery Model\)](#)
- [Example: Piecemeal Restore of Only Some Filegroups \(Full Recovery Model\)](#)
-

```
M:Microsoft.SqlServer.Management.Smo.Restore.SqlRestore(Microsoft.SqlServer.M
anagement.Smo.Server)
```

See Also

[Online Restore \(SQL Server\)](#)

[Performing Piecemeal Restores](#)

[Performing File Restores \(Simple Recovery Model\)](#)

[Performing File Restores \(Full Recovery Model\)](#)

[RESTORE \(Transact-SQL\)](#)

Back Up and Restore of System Databases

SQL Server maintains a set of system-level databases, *system databases*, which are essential for the operation of a server instance. Several of the system databases must be backed up after every significant update. The system databases that you must always back up include **msdb**, **master**, and **model**. If any database uses replication on the server instance, there is a **distribution** system database that you must also back up. Backups of these system databases let you restore and recover the SQL Server system in the event of system failure, such as the loss of a hard disk.

The following table summarizes all of the system databases.

System database	Description	Are backups required?	Recovery model	Comments
master	The database that records all of the system level information for a SQL Server system.	Yes	Simple	Back up master as often as necessary to protect the data sufficiently for your business needs. We recommend a regular backup schedule, which you can supplement with an additional backup after a substantial update.
model	The template for all databases that are created on the instance of SQL Server.	Yes	User configurable ¹	Back up model only when necessary for your business needs; for example, immediately after customizing its database options. Best practice: We recommend that you create only full database backups of model , as required. Because model is small and rarely changes, backing up the log is unnecessary.

System database	Description	Are backups required?	Recovery model	Comments
msdb	The database used by SQL Server Agent for scheduling alerts and jobs, and for recording operators. msdb also contains history tables such as the backup and restore history tables.	Yes	Simple (default)	Back up msdb whenever it is updated.
Resource (RDB)	A read-only database that contains copies of all system objects that ship with SQL Server 2005 or later versions.	No	—	<p>The Resource database resides in the mssqlsystemresource.mdf file, which contains only code. Therefore, SQL Server cannot back up the Resource database.</p> <p> Note You can perform a file-based or a disk-based backup on the mssqlsystemresource.mdf file by treating the file as if it were a binary (.exe) file, instead of a database file. But you cannot use SQL Server restore on the backups. Restoring a backup copy of mssqlsystemresource.mdf can only be done manually, and you must be careful not to</p>

System database	Description	Are backups required?	Recovery model	Comments
				overwrite the current Resource database with an out-of-date or potentially insecure version.
tempdb	A workspace for holding temporary or intermediate result sets. This database is re-created every time an instance of SQL Server is started. When the server instance is shut down, any data in tempdb is deleted permanently.	No	Simple	You cannot back up the tempdb system database.
Distribution	A database that exists only if the server is configured as a replication Distributor. This database stores metadata and history data for all	Yes	Simple	For information about when to back up the distribution database, see Backing Up and Restoring Replicated Databases .

System database	Description	Are backups required?	Recovery model	Comments
	types of replication, and transactions for transactional replication.			

¹ To learn the current recovery model of the model, see [How to: View or Change the Recovery Model of a Database \(SQL Server Management Studio\)](#) or [sys.databases \(Transact-SQL\)](#).

Limitations on Restoring System Databases

- System databases can be restored only from backups that are created on the version of SQL Server that the server instance is currently running. For example, to restore a system database on a server instance that is running on SQL Server 2005 SP1, you must use a database backup that was created after the server instance was upgraded to SQL Server 2005 SP1.
- To restore any database, the instance of SQL Server must be running. Startup of an instance of SQL Server requires that the **master** database is accessible and at least partly usable. If **master** becomes unusable, you can return the database to a usable state in either of the following ways:
 - Restore **master** from a current database backup.
If you can start the server instance, you should be able to restore **master** from a full database backup.
 - Rebuild **master** completely.
If severe damage to **master** prevents you from starting SQL Server, you must rebuild **master**. For more information, see [Rebuilding System Databases](#).



Important

Rebuilding **master** rebuilds all of the system databases.



Related Tasks

- [Create a Full Database Backup](#)
- [Complete Database Restores \(Simple Recovery Model\)](#)
- [Restore the master Database \(Transact-SQL\)](#)

- [View or Change the Recovery Model of a Database \(SQL Server\)](#)
- [Move System Databases](#)

See Also

[Distribution Database](#)

[master Database](#)

[msdb Database](#)

[model Database](#)

[Resource Database](#)

[tempdb Database](#)

Restore the master Database (Transact-SQL)

This topic explains how to restore the **master** database from a full database backup.

Procedures

▶ To restore the master database

1. Start the server instance in single-user mode.

For information about how to specify the single-user startup parameter (**-m**), see [SQL Server Management Studio Tutorial](#).

2. To restore a full database backup of **master**, use the following [RESTORE DATABASE](#) Transact-SQL statement:

```
RESTORE DATABASE master FROM <backup_device> WITH REPLACE
```

The REPLACE option instructs SQL Server to restore the specified database even when a database of the same name already exists. The existing database, if any, is deleted. In single-user mode, we recommend that you enter the RESTORE DATABASE statement in the [sqlcmd utility](#). For more information, see [Using the sqlcmd Utility](#).

Important

After **master** is restored, the instance of SQL Server shuts down and terminates the **sqlcmd** process. Before you restart the server instance, remove the single-user startup parameter. For more information, see [How to: Configure Server Startup Options \(SQL Server Configuration Manager\)](#).

3. Restart the server instance and continue other recovery steps such as restoring other databases, attaching databases, and correcting user mismatches.

Example

Description

The following example restores the `master` database on the default server instance. The example assumes that the server instance is already running in single-user mode. The example starts `sqlcmd` and executes a `RESTORE DATABASE` statement that restores a full database backup of `master` from a disk device: `Z:\SQLServerBackups\master.bak`.



Note

For a named instance, the `sqlcmd` command must specify the `-S<ComputerName>\<InstanceName>` option.

Code

```
C:\> sqlcmd
1> RESTORE DATABASE master FROM DISK = 'Z:\SQLServerBackups\master.bak'
WITH REPLACE;
2> GO
```

See Also

[Performing a Complete Database Restore \(Simple Recovery Model\)](#)

[Performing a Complete Database Restore \(Full Recovery Model\)](#)

[Troubleshooting Orphaned Users](#)

[Detaching and Attaching Databases](#)

[Rebuilding System Databases](#)

[Using the SQL Server Service Startup Options](#)

[SQL Server Configuration Manager](#)

[Back Up and Restore of System Databases](#)

[RESTORE](#)

[Starting SQL Server in Single-User Mode](#)

Backup and Restore: Interoperability and Coexistence

This topic describes backup-and-restore considerations for several features in SQL Server 2012. These features include: file restore and database startup, online restore and disabled indexes, database mirroring, and piecemeal restore and full-text indexes.

In this Topic:

- File Restore and Database Startup
- Online Restore and Disabled Indexes
- Database Mirroring and Backup and Restore

- Piecemeal Restore and Full-Text Indexes
- File Backup and Restore and Compression
- Related Tasks

File Restore and Database Startup

This section is relevant only for SQL Server databases that have multiple filegroups.

Note

When a database is started, only filegroups whose files were online when the database was closed are recovered and brought online.

If a problem is encountered during database startup, recovery fails, and the database is marked as SUSPECT. If the problem can be isolated to a file or files, the database administrator can take the files offline and try to restart the database. To take a file offline, you can use the following [ALTER DATABASE](#) statement:

```
ALTER DATABASE database_name MODIFY FILE (NAME = 'filename', OFFLINE)
```

If startup succeeds, any filegroup that contains an offline file remains offline.



Online Restore and Disabled Indexes

This section is relevant only for databases that have multiple filegroups and, for the simple recovery model, at least one read-only filegroup.

In these cases, when a database is online, the index can be created, dropped, enabled or disabled only if all filegroups holding any part of the index are online.

For information about restoring offline filegroups, see [Online Restore](#).



Database Mirroring and Backup and Restore

This section is relevant only for full-model databases that have multiple filegroups.

Note

The database mirroring feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use AlwaysOn Availability Groups instead.

Database mirroring is a solution for increasing database availability. Mirroring is implemented on a per-database basis and works only with databases that use the full recovery model. For more information, see [Securing Data and Log Files](#).

Note

To distribute copies of a subset of the filegroups in a database, use replication: replicate only those objects in the filegroups you want to copy to other servers. For more information about replication, see [Replication](#).

Creating the Mirror Database

The mirror database is created by restoring, WITH NORECOVERY, backups of the principal database on the mirror server. The restore must keep the same database name. For more information, see [Prepare a Mirror Database for Mirroring \(SQL Server\)](#).

You can create the mirror database by using use a piecemeal restore sequence, where supported. However, you cannot start mirroring until you have restored all the filegroups and, typically, restored log backups to get the mirror database close enough in time with the principal database. For more information, see [Performing Piecemeal Restores](#).

Restrictions on Backup and Restore During Mirroring

While a database mirroring session is active, the following restrictions apply:

- Backup and restore of the mirror database are not allowed.
- Backup of the principal database is allowed, but BACKUP LOG WITH NORECOVERY is not allowed.
- Restoring the principal database is not allowed.



Piecemeal Restore and Full-Text Indexes

This section is relevant only for databases that contain multiple filegroups and, for the simple-model databases, only for read-only filegroups.

Full-text indexes are stored in database filegroups and can be affected by a piecemeal restore. If the full-text index resides in the same filegroup as any of the associated table data, piecemeal restore works as expected.



Note

To view the filegroup ID of the filegroup that contains a full-text index, select the `data_space_id` column of [sys.fulltext_indexes](#).

Full-Text Indexes and Tables in Separate Filegroups

If a full-text index resides in a separate filegroup from all of the associated table data, the behavior of piecemeal restore depends on which of the filegroups is restored and brought online first:

- If the filegroup that contains the full-text index is restored and brought online before the filegroups that contain the associated table data, full-text search works as expected as soon as the full-text index is online.
- If the filegroup that contains the table data is restored and brought online before the filegroup that contains the full-text index, full-text behavior might be affected. This is

because Transact-SQL statements that trigger a population, rebuild the catalog, or reorganize the catalog fail until the index is brought online. These statements include CREATE FULLTEXT INDEX, ALTER FULLTEXT INDEX, DROP FULLTEXT INDEX, and ALTER FULLTEXT CATALOG.

In this case, the following factors are significant:

- If the full-text index has change tracking, user DML will fail until the index filegroup is brought online. Delete operation will also fail until the index filegroup is online.
- Regardless of change tracking, full-text queries fail because the index is not available. If a full-text query is tried when the filegroup that contains the full-text index is offline, an error is returned.
- Status functions (such as FULLTEXTCATALOGPROPERTY) succeed only when they do not have to access full-text index. For example, access to any online full-text metadata would succeed, but **uniquekeycount**, **itemcount** would fail.

After the full-text index filegroup is restored and brought online, the index data and table data are consistent.

As soon as both the base table filegroup and the full-text index filegroup are online, any paused full-text population is resumed.



File Backup and Restore and Compression

SQL Server supports NTFS file system data compression of read-only filegroups and read-only databases.

Restoring files in a read-only filegroup is supported on compressed NTFS files. Backup and restore of these filegroups works essentially as it would for any read-only filegroup, with the following exceptions:

- Restoring a read-write file (including the primary or log files of a read-write database) to a compressed volume fails and displays an error.
- Restoring a read-only database to a compressed volume is allowed.



Note

Log files of read/write databases should never be placed on compressed file systems.



Related Tasks

- [Prepare a Mirror Database for Mirroring \(SQL Server\)](#)
- [Back Up and Restore Full-Text Catalogs and Indexes](#)



See Also

[Back Up and Restore of SQL Server Databases](#)

[Backing Up and Restoring Replicated Databases](#)

[Backup on Secondary Replicas \(AlwaysOn Availability Groups\)](#)