



CHAPTER

3

Availability and Recovery Features

IN THIS CHAPTER:

Protection Against Database or Server Failure

Database Availability Enhancements

Backup and Restore

New improvements in the SQL Server 2005 release have continued to improve SQL Server's availability by increasing its failover clustering capabilities and providing new database options that offer both continuous availability and improved database recovery capabilities. Achieving high availability is relatively easy on a small scale but gets exponentially more difficult as the size of the system increases. SQL Server 2005 provides a number of new features that enhance database availability and recoverability by addressing the primary obstacles that hinder enterprise-level database availability. Some of the factors that have hindered database availability in previous versions of SQL Server include things like delayed failover times, the need to have exclusive database access for selected maintenance operations, and occasional difficulty in getting the server to respond during high CPU usage time, such as when some user's code has gone into an infinite loop. Another factor that can adversely affect database availability is hardware upgrades. Even though hardware upgrades are planned events, they typically require system downtime in order to perform the upgrade. SQL Server 2005 provides a new capability designed to decrease the downtime required by one of the most common hardware upgrades. In this chapter, we'll take a look at these new availability and recovery options found in SQL Server 2005 so that you can understand how these features can be used to implement SQL Server 2005 in a highly available and recoverable production database environment.

Protection Against Database or Server Failure

A database server failure is a breakdown that's caused by either a hardware failure or a software issue that renders the server inoperable for a period of time. Database server failure can also be caused by environmental factors such as a disaster. In this section, you'll learn about some of the most important new features that SQL Server 2005 provides to address database server failures.

Improved Failover Clustering

One key high-availability benefit that SQL Server 2005 derives in part from its support for Windows Server 2003 is significantly improved support for failover clustering. Taking advantage of the enhanced clustering support found in Windows Server 2003, SQL Server 2005 can now be implemented on clusters of up to eight nodes on Windows Server 2003 Datacenter Edition. In addition, SQL Server 2005 supports four-node clustering on Windows Server 2003 Enterprise Edition and

Windows 2000 Datacenter Server. A maximum of two-node clustering is supported in Windows 2000 Advanced Server.

With Windows clustering services, each server in the cluster is called a *node*. All of the nodes in a cluster are in a state of constant communication. If one of the nodes in a cluster becomes unavailable, another node will assume its duties and begin providing the same services as the failed node. This process is called *failover*. Users who are accessing the cluster are switched automatically to the new node.

Clustering can be set up in a couple of basic ways: Active-Active clustering, where all of the nodes are performing work, or Active-Passive, where one of the nodes is dormant until the active node fails. Windows Server 2003 also supports N+I configurations (N active with I spare), which provide a very flexible and cost-effective clustering scenario to enable highly available applications. For example, with an eight-node cluster in an N+I configuration, you can have seven of the eight nodes set up to be available and providing different services while the eighth node is a passive node that can assume the services of any of the seven active nodes. Figure 3-1 illustrates an example eight-node cluster, where seven nodes are active and one node is in standby waiting to step in if any of the seven active nodes fail.

Some of the clustering-specific improvements in SQL Server 2005 include support for an unattended cluster setup. In addition, all of the different services within SQL Server 2005 are fully cluster-aware, including:

- ▶ Database Engine
- ▶ Analysis services
- ▶ Reporting services
- ▶ Notification services

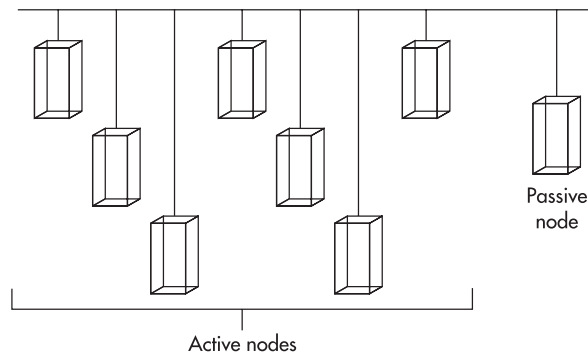


Figure 3-1 Eight-node cluster support

- ▶ SQL Server Agent
- ▶ Full-Text Search
- ▶ Service Broker
- ▶ SQLiMail

In addition, all of the major management tools in SQL Server 2005 are also cluster-aware, including:

- ▶ SQL Server Management Studio
- ▶ Service Control Manager
- ▶ SQL Profiler
- ▶ SQL Query Analyzer

You can find more information about the SQL Server 2005 management tools in Chapter 1.

Database Mirroring

Probably the biggest new feature in the area of availability is SQL Server 2005's support for Database Mirroring. The new Database Mirroring feature protects against database or server failure by giving SQL Server 2005 an instant standby capability. Database Mirroring essentially provides database-level failover. In the event that the primary database fails, the database mirror enables a second, standby database to be available in about 2–3 seconds. Database Mirroring provides zero data loss, and the mirrored database will always be up-to-date with the current transaction that's being processed on the primary database server. The impact of running Database Mirroring to transaction throughput is zero to minimal. The database failover support can be set up to be performed either automatically or manually. The manual failover mode is good for testing, but you'll want your production databases to be able to fail over automatically. Database Mirroring works with all of the standard hardware items that support SQL Server—there's no need for special systems, and the primary server and the mirrored server do not need to be identical. In addition, unlike in high-availability clustering solutions, there's no need for any shared storage between the primary server and the mirrored server. You can see an overview of how the new Database Mirroring feature works in Figure 3-2.

Database Mirroring is implemented using three systems: the primary server, the mirroring server, and the witness. Don't be confused by the names. The primary server is just the name of the system currently providing the database services. All

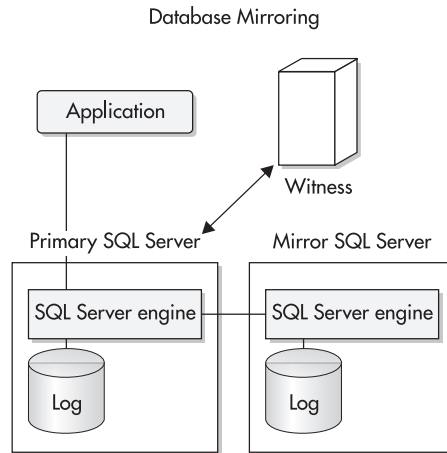


Figure 3-2 Database Mirroring

incoming client connections are made to the primary server. The mirror server's job is to maintain a copy of the primary server's mirrored database. Depending on the configuration of the mirrored server, the primary server and the mirror can seamlessly switch roles. The witness essentially acts as an independent third party, helping to determine which system will assume the role of the primary server. Each system gets a vote as to who will be the primary server. It takes two votes to decide on the primary server. This is important because it's possible that the communications between the primary server and the mirror server could be cut off, in which case each system would elect itself to function as the primary server. The witness would cast the deciding vote.

Database Mirroring works by sending transaction logs between the primary server and the mirroring server. So in essence, the new Database Mirroring feature is a real-time log shipping application. Database Mirroring can be set up with a single database, or it can be set up for multiple databases. When a client system writes a request to the primary server, that request actually gets written to the primary server's log file before being written into the data file, because SQL Server uses a write-ahead log. Next, that transaction record gets sent to the mirroring server, where it gets written to mirroring server's transaction log. After the mirroring server has written the record to its log, it sends an acknowledgment to the primary server that the record has been received, giving both systems the same data in each of their log files. In the case of Commit operations, the primary server waits to receive an acknowledgment from the mirroring server before sending its response back to the client, telling it the operation is completed. In order to keep the data files up-to-date

on the mirroring server, it is essentially in a state of continuous recovery, taking the data from the log and updating the data file.

Database Mirroring is initialized by taking a backup of the database that you want to mirror on your primary server and then restoring that backup to your mirroring server. In other words, the mirrored database must exist on the mirror server before the mirroring process can begin. This puts the underlying database data and schema in place. The backup and restore process can use any of SQL Server's standard media types, including tape or disk. Next, you use the ALTER DATABASE command as shown here to start the mirroring process:

```
ALTER DATABASE <database name> SET PARTNER = '<partner server name>'
```

The ALTER DATABASE command must first be run on the mirroring server, pointing it to the name of the primary SQL Server in the SET PARTNER clause. Then the ALTER DATABASE command is run a second time, this time on the primary server. When the ALTER DATABASE command is run on the primary server, the name of the mirroring server is supplied in the SET PARTNER clause. Once the ALTER DATABASE command has completed on the primary server, database mirroring is set into motion and the primary server will begin shipping logs to the mirroring server. The next step in setting up Database Mirroring is to set up the witness. Much as you set up the primary and mirroring server, you set up the witness by again running the ALTER DATABASE command on the primary server, as shown in the following listing:

```
ALTER DATABASE <database name> SET WITNESS = '<witness server name>'
```

In this case, the database name would be the same database name that was used in the earlier commands. However, this time the SET WITNESS clause is used to specify the witness server. Once the witness server has been set up, Database Mirroring has all of the information that it needs to perform database failover.

Database mirroring essentially gives you a fault-tolerant virtual database. However, that isn't the entire Database Mirroring story. Implementing an instant standby database enables the database to be quickly available after a failure, but what about the user connections to that failed database? That's where the new Transparent Client Redirect feature comes into play.

Transparent Client Redirect

The Transparent Client Redirect feature works very closely with the new Database Mirroring feature to allow client systems to be automatically redirected to the mirroring server when the primary server becomes unavailable. This new feature is implemented in

the new SQL Server 2005 Microsoft Data Access Components (MDAC), and no changes are required to the client- or data-layer applications. The MDAC middleware is aware of both the primary and mirroring servers. MDAC acquires the mirroring server's name upon its initial connection to the primary server. When the connection to the primary server is lost, MDAC will first try to reconnect to the primary server. If that first connection attempt fails, then MDAC will automatically redirect its second connection attempt to the mirroring server. Just as with clustering, if the connection is lost in the middle of a transaction, then that transaction will be rolled back and must be redone after the client connects to the mirroring server.

When to Use Clustering or Database Mirroring

Both clustering and database mirroring are enterprise-ready solutions that are capable of providing a backup system in case of database or server failure. They have many similarities. Both are capable of enabling zero data loss, both have automatic failure detection, both provide automatic failover, and both enable client reconnection. However, there are some important differences. As its name implies, Database Mirroring works at the database level, while clustering works at the server level. Database Mirroring provides nearly instant 2–3 second failover time, while clustering typically has about a 30-second failover time—sometimes longer, depending on the level of database activity and the size of the databases on the failed server. Database Mirroring also protects against disk failures slightly better because there is no shared disk storage, as there is in a clustering solution. There is virtually no distance limitation for Database Mirroring, while clustering has a limit of about 100 miles, because there is a time limit for transmitting the heartbeat between cluster nodes. In addition, Database Mirroring is a simpler technology that's easier to implement than clustering. On the other hand, Database Mirroring can't be used for the system databases, while clustering does protect the system databases as well as the user databases. Basically, clustering is the better solution for protecting an entire server, while Database Mirroring is the better solution for protecting a single critical database or application.

Database Availability Enhancements

Although the previous scenarios all address important factors that reduce database availability, there's probably no single factor that affects database availability and recoverability more than database maintenance. While it's certainly possible to go for years without a database failure, database maintenance is a daily issue that you can't avoid. SQL Server 2005 tackles the challenges of this vital area head-on with the addition of two new features: Database Snapshots and early restore access. In the

next part of this chapter, we'll look at each of these new availability and recoverability features in more detail.

Database Snapshot

The new Database Snapshot feature provides a read-only snapshot of a database at a specific point in time. A Database Snapshot is best suited either for creating copies of a database for reporting or for creating a backup copy of the database that you can use to revert a production database back to a prior state. When a Database Snapshot is created, the system makes a metadata copy of the specified database at that particular point in time. Any subsequent changes that are made to the original database are not reflected in the Database Snapshot. Applications connect to a Database Snapshot exactly as if it were another database. A Database Snapshot can be created for any database. Database Snapshots are created as part of the CREATE DATABASE DDL statement. You can see how to create a view on the example AdventureWorks database in the following listing:

```
CREATE DATABASE AdWSnapShot_080104_0700 ON
( NAME = AdventureWorks_Data,
  FILENAME = 'C:\Program Files\Microsoft SQL Server
  \MSSQL.1\MSSQL\DATA\AdventureWorks_data_040422_1800.ss')
AS SNAPSHOT OF AdventureWorks;
```

In this example, you can see that a Database Snapshot named AdWSnapShot_080104_0700 is created on the AdventureWorks database. Database Snapshots are created using the AS SNAPSHOT OF clause that you can see in the listing. When you create a Database Snapshot, you must specify all of the data files that are used in the source database. Since the AdventureWorks database consists of a single data file, only the AdventureWorks_Data file must be specified in the CREATE DATABASE statement. You do not specify the log files.

Creating a Database Snapshot is an inexpensive operation server-wise, as the server basically uses metadata in conjunction with recovery to create the viewpoint. It's important to understand that Database Snapshots are not a complete copy of a database. Instead, creating a Snapshot is a metadata-only operation. A Database Snapshot uses the same data pages as the original database, so it doesn't require a great deal of additional disk space. Database Snapshots are built using copy-on-write technology where anytime a change is made to one of the source database's data pages, a copy of that page is saved for the Database Snapshot and then the updated page is written the same way it normally would be. When the Database Snapshot is accessed, it uses the shared data pages until it gets to the changed page, and then it

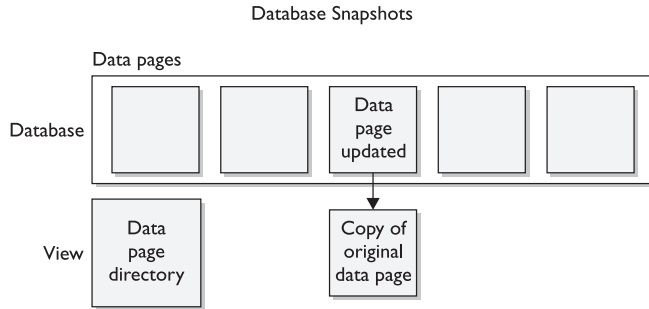


Figure 3-3 Database Snapshots

will look at the pages that have been copied rather than the data pages that contain the updated data. In this way, the Database Snapshot needs storage for only those pages that have been changed since the time the Database Snapshot was created. Figure 3-3 illustrates how Database Snapshots work.

Database Snapshots can be combined with Database Mirroring to create a reporting server based on the data that's on the mirrored server (Figure 3-4). Normally, the data on the mirrored server is always in recovery mode, so it can't be accessed by an application. However, you can create a Database Snapshot that's based on the mirrored database and that Database Snapshot can be accessed in read-only mode for reporting.

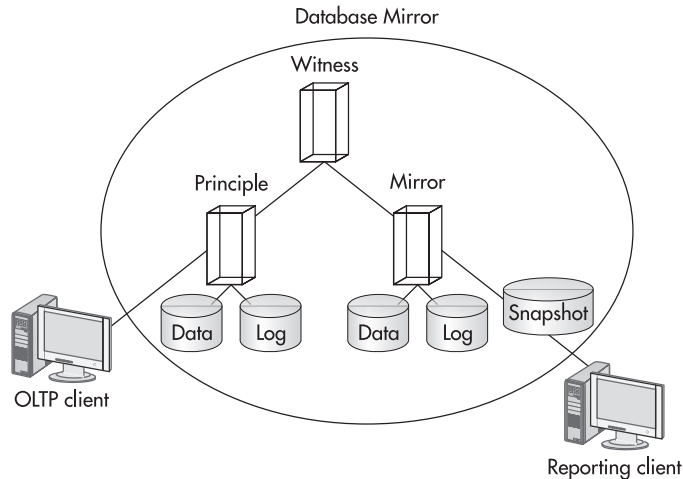


Figure 3-4 Database Mirroring and Database Snapshot create a reporting server

Although the database on the mirroring server can't be directly accessed because it's in an ongoing state of recovery, that doesn't affect the Database Snapshot, which accesses a snapshot of the data pages in the mirroring server's database. Creating a Database Snapshot on the mirroring server enables you to better utilize the processing power of the mirroring server by enabling you to shift your static reporting to that server. One thing to consider when using Database Snapshots on a Database Mirror is that in the event of a failover, the existing Database Snapshots that were created on the mirroring server will remain intact.

It's important to realize that Database Snapshots are an availability feature, not a failover feature. They provide more ways to access your data, increasing your data's availability. They are not a failover feature—if the original database is unavailable, the Database Snapshot will also be unavailable.

Early Restore Access

Every time the SQL Server database is started or a database is restored, that database must go through a period of recovery where any transactions that are in the log are applied to the data files. The phase of recovery is often referred to as redo. As soon as the redo portion of the recovery process is complete, then all of the undo operations are performed as any incomplete transactions in the log are rolled back. With SQL Server 2000, the database was not available until all of the redo and undo operations were completed. If a database was brought down while it was active and that database was quite busy, then there could be a somewhat lengthy delay before that database was available again, as you needed to wait until all of the entries in the log were processed.

SQL Server 2005's new Early Restore Access enables the databases to become available immediately after the redo portion of the recovery process is completed. With SQL Server 2005, when the database is restarted, all of the open transactions that are in the log are redone and then the database is immediately available. The net result is that the database is available much sooner. For the transactions that weren't committed, SQL Server still holds the locks on the data pages used by those transactions so that the transactions will remain consistent even though the database is in use. SQL Server 2005 then begins the process of undoing these transactions while the database is active. Users can initiate read/write operations to the database during the undo phase of recovery. However, any attempts to access the data that SQL Server has locked during the undo process will experience blocking until the undo process releases the locks on that data. Figure 3-5 illustrates the difference in availability between SQL Server 2000 and SQL Server 2005.

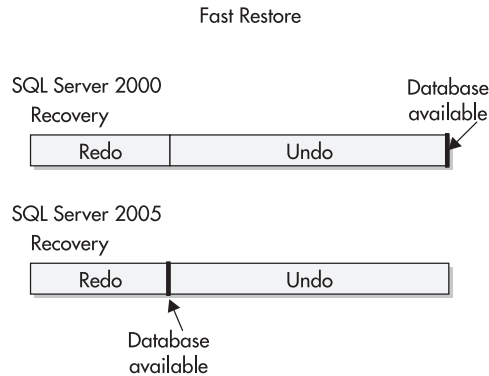


Figure 3-5 *Early restore access*

The RESTORE action is now very granular and defines the scope of the redo option. You can restore full, partial, or filegroup backups. If you indicate that you want to restore a filegroup, then only that filegroup's data is added to the roll-forward operation. If you indicate that you want to restore a full backup, then all of the data in the backup set will be used for the redo operation.

Online Index Operations

With prior versions of SQL Server, when an index was being rebuilt you couldn't perform any update operations on the table until the index rebuild had finished. SQL Server 2005's new online index operations feature extends SQL Server's availability by enabling applications to update, insert, and delete rows from the table while the index is being rebuilt. The new online index feature performs this magic by keeping two copies of the index: one that the applications can continue to use and a second temporary index that's used while the index is being rebuilt. The SQL Server engine maintains both indexes with any changes as the rebuild is being performed. When the rebuild is finished, the old index is dropped and replaced with the new index. Online index rebuild is supported for CREATE INDEX, ALTER INDEX REBUILD, ALTER INDEX DISABLE, DROP INDEX, ALTER TABLE ADD CONSTRAINT, and ALTER TABLE DROP CONSTRAINT statements. Although SQL Server 2005 now supports online index rebuild, it does come at the cost of additional overhead, so you can still choose to rebuild your indexes offline.

The following listing illustrates how the new online indexing feature is used:

```
CREATE INDEX MyIndex ON Person.Contact (LastName) WITH (ONLINE=ON)
```

Here, you can see that the standard CREATE INDEX statement is used to create an index named MyIndex on the LastName column of the table named PersonContact that's in the AdventureWorks database. The ONLINE=ON clause enables the index to be created online.

Fine Grained Online Repairs

Another new feature that enhances the availability of SQL Server 2005 is the ability to perform fine-grained restores. While not the table-level restores that some people have looked for since that feature was removed after SQL Server 6.5, the new fine-grained restore feature in SQL Server 2005 enables you to restore select filegroups in a database while the remainder of the database continues to be available. For SQL Server 2000, that basic unit of availability is the database. All of the components of the database must be intact before that database is available. With SQL Server 2005, the unit of availability is now the filegroup. SQL Server 2005 enables you to restore a filegroup at a time or even a page or group of pages at a time, and the rest of database can continue to be available as long as the primary filegroup is up.

Damaged Page Tracking

A closely related new feature that ties in with SQL Server 2005's ability to perform page-level restores is the ability to track damaged pages. Any bad pages that are encountered on a read operation are tracked in a table, and by using the fine-grained restore capability, you can restore on a page-by-page basis while the database remains online. Any transaction that uses data from a damaged page is rolled back. If the bad page happens to turn up during a transaction rollback, then the database will be forced to restart.

Dedicated Administrator Connection

Another new feature found in SQL Server 2005 that helps to provide better availability is the new dedicated administrator's connection. The dedicated administrator's connection provides the DBA with access to the server regardless of the server's current workload. This enables the DBA to access to the server and kill any runaway processes. To start the new SQLCMD tool in dedicated administrative mode, you would enter the following command:

```
C:\Sqlcmd -A
```

More information about using the dedicated administrator's connection appears in Chapter 1.

Hot-Plug Memory

Another new availability feature that's derived from Windows Server 2003 is Hot-Plug memory. Hot-Plug memory allows you to add RAM while the system is running. As you might expect, this feature requires support from the underlying OS and the hardware platform. In the case of SQL Server 2005, this means that SQL Server must be running on Windows Server 2003 to be able to take advantage of this availability feature. If the platform supports this feature, you can add memory on the fly and SQL Server 2005 will be able to dynamically recognize the additional RAM without requiring a server reboot or any downtime. While Hot-Plug memory allows you to dynamically add RAM on the fly, it doesn't allow you to remove it.

Improved Dynamic Configuration

All configuration within SQL Server 2005 is now dynamic, including CPU and I/O affinity. With SQL Server 2005, you can now change these values and the effects will take place immediately. There's no longer any need to restart the server. In addition, changes to Address Windowing Extensions (AWE) are also all dynamic. SQL Server 2005 can automatically adjust to changes to the physical AWE size dynamically. This feature is designed to work in conjunction with the ability to use Hot-Plug memory on the fly. This feature requires Windows Server 2003.

SQL Server Transaction Isolation Levels

Another area that affects database recoverability is the transaction isolation level that is used by the application. SQL Server 2005 provides a new level of transaction called Snapshot Isolation, giving it added flexibility for data access. The standard four ANSI isolation levels that were supported on earlier version of SQL Server (Serializable, Repeatable Read, Read Committed, Read Uncommitted) all protect transactions from one another by taking locks on the underlying data rows so that other applications can't change the data. These transaction levels all continue to be supported. The new Snapshot Isolation provides increased data availability for read applications. With Snapshot Isolation, SQL Server 2005 performs a type of optimistic locking where SQL Server doesn't take any locks on the rows involved in a transaction. Instead, it keeps track of the row's state when the transaction is opened. When a Snapshot transaction is opened, the system essentially copies the row data for the transaction, enabling your application to continue to see the row data as it was at the time the transaction was started. No locks are placed on the rows. This enables non-blocking consistent reads in an OLTP application. Because Snapshot Isolation doesn't hold any locks, data writers don't block readers and

readers don't block writers. The new Snapshot Isolation level is really meant for applications that have an emphasis on read operations. When your application reads the rows in a transaction, it may be reading old data because other applications are able to change the data. While this new isolation level does allow writes, there is additional overhead in all updates. Snapshot Isolation is useful when you need a time-consistent view of all of the data in your database. For update applications, Snapshot Isolation is best used in scenarios where the cost of locking the data outweighs the cost of occasionally rolling back a transaction.

Backup and Restore

SQL Server 2005 also sports several new features in the area of backup and restore technology. While many of the previous availability features were related to some of these new backup and restore features—particularly fine-grain online operations—this section focuses primarily on those changes that Microsoft has made to make the backup and restore process more robust and easier to implement.

Partial Restore

With SQL Server 2000, the database was not available during a restore operation. With SQL Server 2005, the database is online as soon as the primary filegroup is restored. Only the data being restored is unavailable. If a user accesses data that's found in the primary filegroup, the operation will succeed with no indication to the user that the rest of the database is still being restored. If the user does attempt to access data from a filegroup that is currently being restored or that is still offline, then the database will return a message that the data is offline.

Media Reliability Enhancements

SQL Server 2005 also provides a number of enhancements in the way that it deals with media. In particular, it now allows backups to be performed to mirrored devices, it provides enhanced checksum integrity, and it enables a restore operation to continue even if errors are encountered.

Backup Media Mirroring

One of the new media reliability features found in SQL Server 2005 is the ability to support media mirroring. Media mirroring enables you to simultaneously perform a

backup to multiple backup devices. For instance, you can back up your database to both tape1 and tape2 at the same time, obtaining two identical copies of the backup set. Redundant backup media provide an important safeguard that can help protect your organization from media errors and help to ensure the ability to perform a successful restore. The following example illustrates using the new media mirroring feature:

```
BACKUP DATABASE AdventureWorks TO
    TAPE='\\.\tape1'
MIRROR TO
    TAPE='\\.\tape2'
WITH FORMAT, MEDIANAME = 'ADWBackup'
```

This example shows the AdventureWorks database being backed up to tape1 and mirrored to tape2. Performance of the backup is not affected by adding a backup mirror. Up to four mirrors can be applied to a single backup.

Improved Verification of Backups

With SQL Server 2000, using the RESTORE VERIFYONLY command just caused SQL Server to read the tape without going any farther into the restore process. Using the RESTORE VERIFYONLY command in SQL Server 2005 does everything short of actually restoring the data in the restore media, giving you a much better idea of the viability of the data in the backup set.

Continue Past Restore Errors

Another new media reliability enhancement is the ability of the restore operation to continue past media errors. Prior versions of SQL Server would abort a restore operation if any error was encountered during the restore process. However, in cases where this was the only media version available, this could be a serious impediment to recovering whatever data was available. With this new feature, SQL Server 2005 will allow the restore process to continue as far as possible, ignoring the media errors it encounters. After the restore process has finished, the database can be manually repaired.

Database Page Checksums

The new Database Page Checksum feature enables the database to detect disk and I/O errors that are not reported by the disk subsystem. When the Database Page Checksum feature is enabled, SQL Server will calculate a checksum value as the page is written to disk and write that checksum along with the data. When the page

is read, another checksum is calculated and compared to the original checksum written with the data. If they are different, an error is reported. The following command shows how Database Page Checksums can be added to an existing database:

```
ALTER DATABASE AdventureWorks  
    SET PAGE_VERIFY CHECKSUM
```

Concurrent Database and Log Backups

Another new feature in the SQL Server 2005 backup area is the ability to perform database backups and log backups concurrently. Using SQL Server 2000, you had to wait to back up the log until after the database backup had completed. With SQL Server 2005, database backups no longer block log backups. Likewise, you can also back up files and filegroups at the same time that you back up the transaction log. However, you are limited to performing one data file backup at a time per each database.

Backup of Full-Text Catalogs

One other backup enhancement that's part of SQL Server 2005 is the ability to back up Full Text Catalogs. With SQL Server 2000, Full Text Catalog data was maintained outside of SQL Server by the Microsoft Search Service. Backing up the SQL Server database that contained full-text data didn't automatically result in the full-text catalog being backed up as well. This opened up the possibility of the catalog being out-of-sync with the full-text data being restored. SQL Server 2005 fixes this problem by including the ability to automatically back up all of the external full-text catalogs at the same time that the database is backed up. This ensures that the full-text catalog and the data remain consistent between the backup and restore operations. The DATABASE ATTACH and DEATTACH commands also have an option to include any full-text catalogs. Links in the database point to the external files used. When the database is backed up, these external files are also backed up, ensuring database consistency.