Achieve High Availability for SQL Server

Zach Nichter

**At a Glance:**

* Mirroring
* Database snapshots
* Log shipping
* Clustering
* Replication

Download the code for this article: [NichterHA2007\_03.exe](http://download.microsoft.com/download/f/d/0/fd05def7-68a1-4f71-8546-25c359cc0842/NichterHA2007_03.exe) (151KB)

High availability is a concept that every database administrator should understand. It refers to the responsiveness and accessibility of a system. Sometimes high availability means a response time of seconds while other situations require response times in

fractions of a second. I once consulted at a company where Web servers required SQL queries to have round-trip response times in milliseconds; if the response exceeded that limit, the database system was considered down and the Web server would reconnect to the next available database server.

With users demanding faster and faster applications, knowing how to achieve high availability and quick response times will help you plan your data-dependent applications wisely.

Fortunately SQL Server™ 2005 has a number of options to improve availability, including replication, clustering, database mirroring, database snapshots, and database log shipping. I'll take a look at these features and provide some insight into how you can decide which are right for your environment. Let's start with **Figure 1**, which describes availability options in SQL Server 2005.

http://i.msdn.microsoft.com/Global/Images/clear.gif Figure 1 High-availibility options

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Technology Attributes | Replication | Database Mirroring | Clustering | Database Snapshots | Log Shipping |
| High-availability option |  |  |  |  |  |
| Tolerate high transaction requirements |  |  |  |  |  |
| Real-time data availability |  |  |  |  |  |
| Data image is read-only |  |  |  |  |  |
| Unique hardware configuration |  |  |  |  |  |
| Low cost |  |  |  |  |  |
| Provides data recovery |  |  |  |  |  |
| Automated failover |  |  |  |  |  |
| Potentially complex implementation/management |  |  |  |  |  |
| Possible performance considerations |  |  |  |  |  |

**Defining High Availability**

One of your first objectives when planning a high-availability application is defining what it means in your particular environment. To some organizations, high availability means that there must be redundant hardware equal to the production hardware, requiring both the data and the hardware to have an up-time and availability of 99.995 percent or higher. Other organizations might require only the data itself to be highly available, with less concern for production-level performance should a failover be necessary. Defining high availability is important in determining the right solution for your situation.

You also need to identify the types of outages you may experience and indicate how they affect your Service Level Agreements (SLA). Outages that can affect your availability include planned, unplanned, and degraded performance.

A planned outage is usually a scheduled maintenance window about which the users of the systems are informed in advance. An unplanned outage is generally the result of a hardware or software failure that causes your data to become inaccessible. Performance degradation can also cause outages and is often measured in end-user response time, which is typically agreed upon in advance by the business as well as the IT organization in some form of SLA.

In addition to identifying potential sources of outages, you must also determine the activity level of your data and whether it must always be online or can be near-line or off-line at times. You must also decide if the availability option is going to be in the same geographic location or a remote location. Budget limitations will likely play a role in your decision as well. Now let's look at each availability option.

**Database Mirroring**

Before I dig into database mirroring, I should define the terminology.

**The principal** is the primary production server housing the database that is sending its transaction logs continuously to the mirror server and database.

**The mirror** is the secondary server that houses the backup copy of the database. The mirror copy is consistently synchronized with the principal database.

**The role** indicates the purpose of the particular server-whether it serves as the principal or mirror.

**The witness** is the instance that monitors the principal and mirror servers as they perform their duties and can initiate an automatic failover.

**A partner** can be either the principal server or the mirror server.

In a typical environment, the principal database will be backed up on the principal instance and the backup will be restored on the mirror instance (see **Figure 2**). Once the database is restored, mirroring must be set up on the principal server either using the properties window of the principal database in the SQL Server Management Studio (SMSS) or by using T-SQL scripts.

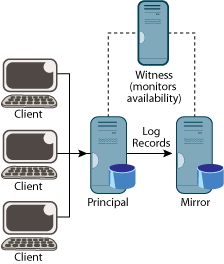


Figure 2**Database mirroring architecture**(Click the image for a larger view)

Once mirroring is configured and the mirroring session has been established, the principal and mirrored databases will synchronize. The principal database will then send its transaction log of events that occurred since the time the last backup was applied on the mirror. The mirror will receive the log and attempt to apply it as quickly as possible. If you're using SQL Server 2005 Enterprise Edition, this process is multithreaded; otherwise it is a single-threaded operation. As soon as these logs are applied to the mirror, the databases are considered synchronized and will remain synchronized until the mirrored session is broken.

As clients execute new transactions, the principal server executes the transactions against the principal database and while doing so ships the transaction log records from the principal database to the mirrored database's redo log, or log queues, where it is picked up and applied to the mirrored database. Once the transaction is applied and committed on the mirror database, a reply is sent to the principal announcing that the transaction has been committed on the mirror. The principal will not confirm any new transactions that might come into the system until the acknowledgment is received from the mirror.

In the case of a failure, the mirror can initiate an auto-failure and the witness supports the process by determining whether the principal database is available. Once a failure occurs, the problem on what is now the mirror partner must be resolved before it can again become the principal partner. After the issues are resolved on the mirror partner, the move back to the mirror partner server is initiated and the databases synchronize again. Once they are in sync, the mirroring session can begin again.

This particular mirroring mode is a high-safety mode that provides synchronous transactional operations by turning on transactional safety, but it does not require a witness because it does not take advantage of automated failover; all failovers are manually initiated. There is another type of mirroring mode that provides synchronous transactional operations-the high-availability mode. It requires not only that transactional safety be turned on, but also that a witness be used for automated failover in the case of a failure.

The third and final mode available in mirroring is high-performance. It requires that transactional safety be turned off allowing for asynchronous operational support which, in turn, allows the transactions on the principal partner to commit without having to wait for the transaction record to be written to the mirror. High-performance mode does not require a witness in the configuration.

Note that mirroring requires the same edition of SQL Server on principal and mirror but not on the witness, which can be SQL Server Express Edition. In addition, it is important that the principal database be in full recovery mode.

ADO.NET 2.0 is integrated with SQL Server 2005 and includes the ability to support database mirroring and provide a transparent failover for the application to the mirrored environment. This gives your ADO.NET application a way to automatically fail over without any additional coding or configuration if a connection to the principal database cannot be made. The configuration is as easy as specifying a common user between the two environments and the failover partner in the connection string. The following is an example of an ADO.NET connection string identifying the failover partner for the mirrored database environment:

[Copy Code](javascript:CopyCode('ctl00_MTContentSelector1_mainContentContainer_ctl03_code');" \o "Copy Code)

"Provider=SQLNCLI.1;Data Source=MirrorDB;Failover Partner=SQL03;

Initial Catalog=AdventureWorks;

Persist Security Info=True;User ID=TestUser; Password=TestPswd; Pooling=True;

Connect Timeout=5;Application Name=ADOMirrorTest"

Database mirroring might be a good option for your company depending on your application and data requirements, but there are a number of things you need to consider for optimal performance. For instance, how high is the transaction rate of the system or the volume of data change? When you consider database mirroring, it is critical to determine whether bandwidth and network speed are sufficient to handle the volume of data and the transaction processing rate. Also, consider the saturation of the link. This is especially important if the mirror is in a different geographic location. Monitoring your system beforehand is crucial in determining if there are any environmental limitations that would prevent mirroring from operating efficiently.

Database mirroring can be a particularly good option if you are trying to keep costs down. In fact, the database mirroring architecture requires no shared disks and no advanced or specialized skill to run the environment. And unlike clustering, database mirroring does not require that both of the partners have the same hardware. Furthermore, it's simple to implement mirroring using the setup wizard found on the mirroring tab of the database properties window (see **Figure 3**). I also recommend reading the whitepaper "Database Mirroring Best Practices and Performance Considerations" at go.microsoft.com/fwlink/?LinkId=80897 for more helpful information.

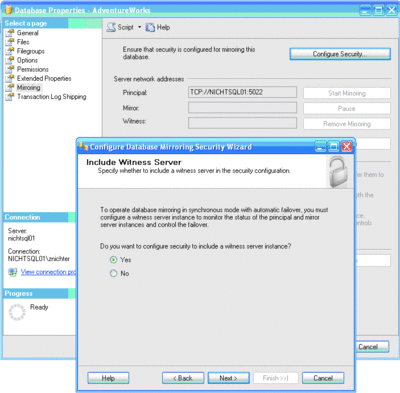


Figure 3**Mirroring setup wizard**(Click the image for a larger view)

**Database Snapshots**

Database snapshots are a new technology offered in SQL Server 2005 Enterprise Edition, but are not considered a high-availability option. Database snapshots should be used as a recovery or a viable reporting option when used with other technologies. A snapshot is simply a read-only view of a database at a particular point in time.

A snapshot is created by using a CREATE DATABASE command like so:

[Copy Code](javascript:CopyCode('ctl00_MTContentSelector1_mainContentContainer_ctl05_code');" \o "Copy Code)

CREATE DATABASE SnapDB\_20061028\_2030 ON

(NAME = SnapDB\_Data, FILENAME =

'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\SnapDB\_20061028\_2030.snp')

AS SNAPSHOT OF SnapDB;

GO

When the database snapshot is created, it uses one or multiple files called sparse files rather than data files as a typical database would. These sparse files are essentially virtual holding places that initially consume no data. They are used to hold data only if the data it is changed or deleted in the source database. Data is written to the sparse files one data page at a time and the actual database snapshot only displays data that has been changed since the snapshot was taken. The rest of the data comes from the source database's data pages.

The database snapshot files are allocated at the size of the database when the snapshot was taken. The allocated size does not tell you how much data they actually hold. To get that information, run a T-SQL statement like this one:

[Copy Code](javascript:CopyCode('ctl00_MTContentSelector1_mainContentContainer_ctl06_code');" \o "Copy Code)

SELECT \*

FROM fn\_virtualfilestats(DB\_ID(N'SnapDB\_20061028\_

2030'), 1);

GO

Because of the way data is stored in sparse files and the source database, when a database snapshot is accessed, the data pages in the original database data files and data pages from the sparse files of the snapshot are retrieved. Snapshots can only live on the server with the source database that the snapshot was taken from because of the need to share data pages. Because this architecture does not alleviate I/O on the source database, snapshots are not a valid reporting option to pursue as they will not represent the true state of the database.

Consider a scenario where database mirroring is used along with snapshots. This allows for the reporting data, the mirror, and snapshot databases to be separated physically from the principal database. The database snapshot is scheduled using the SQL Server agent and a custom script to provide refreshed snapshots at regular intervals. The example script in **Figure 4** shows the stored procedure used to accomplish this. It is designed to be used inside of a job to manage creating and dropping snapshots for a particular database in your environment. This would allow for an acceptable reporting solution because the reporting data would be isolated from production data.

http://i.msdn.microsoft.com/Global/Images/clear.gif Figure 4 Stored procedure for scheduling snapshots

[Copy Code](javascript:CopyCode('ctl00_MTContentSelector1_mainContentContainer_ctl14_ctl00_ctl00_code');" \o "Copy Code)

use msdb;

GO

set nocount on

GO

CREATE PROCEDURE usp\_snaprefresh

@database sysname = NULL --name of the database to snapshot

,@keepsnap int = 24 --# of hours to keep a snapshot

--after it was created

--use a value of '0' to keep all

--existing snapshots

,@fileloc sysname --location for snapshot

= 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\'

AS

DECLARE

@dt datetime ,@cnt int

,@databaseid int ,@snap sysname

,@sql nvarchar(1000) ,@yy varchar(4)

,@mm varchar(2) ,@dd varchar(2)

,@h varchar(2) ,@m varchar(2)

,@lname sysname ,@pname sysname

,@file sysname ,@pos1 int

,@pos2 int

-- initialize variables

SELECT

@dt = getdate()

,@cnt = 0, @pos1 = 0, @pos2 = 0

,@databaseid = db\_id(@database)

-- check if valid database was provided

IF @databaseid IS NULL

BEGIN

RAISERROR ('Missing database name. Rerun the procedure specifying a

valid database name.',16,1)

RETURN (0)

END

--determine if snapshots should be kept

IF @keepsnap <> 0

BEGIN

-- determine if other snapshots exist for this server older than @

-- keepsnap value hrs

SELECT ROW\_NUMBER() OVER(ORDER BY name DESC) AS [RowNum],

name INTO #t1

FROM master.sys.databases

WHERE source\_database\_id = @databaseid

AND create\_date < dateadd(hh,-(@keepsnap),getdate())

IF (SELECT max(RowNum) FROM #t1 where name is not null) > 0

BEGIN

WHILE @cnt <= (SELECT max(RowNum) FROM #t1)

BEGIN

SELECT @snap = name FROM #t1 WHERE RowNum = @cnt

PRINT 'Dropping snapshot ''' + @snap + ''''

SET @sql = 'DROP DATABASE ' + @snap

EXEC(@sql)

SELECT @cnt = @cnt + 1

END

END

END

-- break apart point in time date time information for file name

SELECT @yy = convert(varchar(4),year(@dt)),@mm = convert(varchar(2),

month(@dt))

,@dd = convert(varchar(2), day(@dt)),@h = convert(varchar(2),

datepart(hh,@dt))

,@m = convert(varchar(2), datepart(mi,@dt))

-- piece together the database snapshot name and the file name

SELECT @file = @database + '\_' + @yy + @mm + @dd + '\_' + @h + @m

-- identify logical file name of primary data file

SET @sql = 'SELECT name INTO tempdb..t1

FROM ' + @database + '.sys.database\_files

WHERE file\_id = 1'

EXEC(@sql)

--setting logical filename for the snap

SELECT @lname = name FROM tempdb..t1

-- making sure the file location ends with '\'

IF substring(@fileloc, len(@fileloc), 1) <> '\'

BEGIN

SET @fileloc = @fileloc + '\'

END

-- build sql statement to be run

SET @sql = N'CREATE DATABASE ' + @file + ' ON

(NAME = ' + @lname + ', FILENAME = ''' + @fileloc + @file + '.snp'')

AS SNAPSHOT OF ' + @database

EXEC(@sql)

-- cleanup

DROP TABLE tempdb..t1;

GO

Keep in mind, though, that database snapshots are considered temporary because snapshots cannot be backed up and cannot exist without their source database. If a sparse file runs out of space, the snapshot will be considered corrupt and must be dropped.

In addition, with database snapshots system performance can be degraded during data modification operations on the source database because the data pages are being written to the sparse file for each snapshot from the source database data files. This in turn multiplies the number of writes by the number of snapshots a database has.

Read permissions are defined by the source database at the point of the snapshot and cannot be changed. The snapshot must reside in the same instance as the source database because they share data pages as well as the same buffer cache.

You should consider using snapshots as a reporting solution only if you are also using mirroring; otherwise there is no performance gain for your environment. Database snapshots are possibly the fastest way to preserve data before a questionable operation is run on a system. A database can be reverted back to the state of the snapshot or the data can be pulled out of the snapshot to replace data in the source database.

You'll need to determine a naming standard for the snapshot databases. The standard that I use is originaldatabasename\_date\_time.snp. It specifies the source database first, then the day and the time (in 24-hour format) that the snapshot was taken.

**Log Shipping**

Log shipping is a limited high-availability option that utilizes backup and recovery to establish a very inexpensive solution. Log shipping takes advantage of transaction log backups on a scheduled interval to keep a secondary database up to date.

Log shipping in SQL Server 2005 uses wizards to walk through the process of setup, scheduling, initialization, and monitoring (see **Figure 5**). The process is simple and can be completed in just minutes.

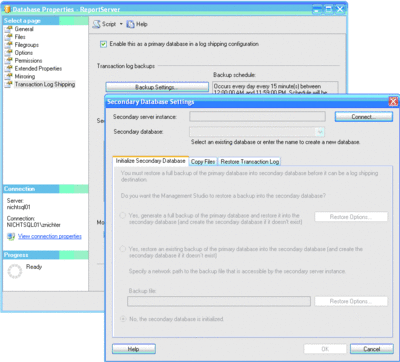


Figure 5**Log shipping setup wizard**(Click the image for a larger view)

After the primary database is identified, a schedule is created and the file age determined for backup files. Next, the file shares for the backup files must be established. After the file shares are set up, the secondary database file location needs to be established and the database must be initialized by completing a restore of the primary database. Finally the schedules must be set for the backup file copies and the restores of the transaction log backups along with any alerts or delays needed for each step.

Once setup is complete, log shipping backs up the transaction log of a database on a scheduled basis to a shared network location. After the files are sent to the share, the backup is applied to the secondary database on a set schedule.

Log shipping works well in many situations because of its simplicity. It is a good option for high availability that is inexpensive and can handle a high transaction system. The secondary database that is employed in log shipping can be used in read-only mode, which makes it handy for a reporting database. Log shipping requires minimal overhead but does necessitate an alert policy in case of handling failures.

A couple of things to consider if you're thinking about using log shipping in your environment are its simple setup and management and the fact that it requires little to no specialized skill. Log shipping is not a real-time high availability solution, however, though it can be combined with database mirroring for this purpose. It is limited by schedules as well as the operations such as backups, file copies, and restores. It also requires manual failover. For these reasons, log shipping provides a simple solution for the environment that is not quite as sensitive to time requirements.

**SQL Server Clustering**

Server clustering works at the OS level and involves duplicate hardware as well as shared disk resources for the cluster to access. Clustering is perhaps the least intrusive to end users, but it is also likely to be the most costly. It requires at least twice the amount of hardware that's needed to run a non-clustered instance.

The topic of clustering is quite involved, so rather than exploring all the details here, I'll give you a general overview. Clustering requires two or more servers, all of which must be installed with the same version of Windows® 2000 Advanced or Datacenter editions, or Windows Server® 2003 Enterprise or Datacenter editions. It also requires the installation of Microsoft® Cluster Services (MSCS) which handles ownership of shared resources between servers and manages IP addresses, shared disks, and network names. Clustering also requires a shared disk resource, usually in the form of a Storage Area Network (SAN) or SCSI attached storage.

A SQL Server instance is also considered a resource and both Standard and Enterprise editions of SQL Server 2005 can be installed in a cluster configuration. See the "Feature Comparison Chart for SQL Server 2005" (microsoft.com/sql/prodinfo/features/compare-features.mspx)for a list of features supported by both editions of SQL Server 2005.

Once the resources are established on the cluster, the secondary node of the cluster keeps in regular communication with the primary node of the cluster through a heartbeat established on a private network between the two nodes of the cluster. The heartbeat is an interval checkpoint that is taken to determine if the primary has failed.

In the case of a failure of the primary, the resources are moved to the secondary node while keeping the state of the logical server in place. This allows the clients to continue to work with only a pause in interaction. The entire failover process can take anywhere from 5 seconds or less to 30 seconds or more in some cases depending on the hardware, software, and networking components involved in the cluster.

Clustering can be an expensive, complex technology requiring specialized skill to resolve failures on the system; however, it provides the smoothest failover for end users of any of the automated failover options. Each application is different and some may not be cluster-aware or compatible, which in a worst-case scenario requires the application to reconnect.

**Replication**

SQL Server 2005 replication can also be used in high-availability architectures. It offers four types of replication: snapshot, transactional, peer-to-peer, and merge. Peer-to-peer is actually just a form of transactional replication so I won't discuss it here.

With replication you get the option of having a secondary site and database for high availability that is every bit as functional as the primary database. This can be achieved using merge replication, which takes transactions from both the primary and the secondary databases and merges the changes from both into one another. As you can guess, a conflict resolution procedure is needed in this configuration.

Transactional replication is very similar in logical design to database mirroring. Transactions that are applied to the primary database are shipped to the secondary database to ensure that the environment remains consistent. Once the transactions arrive they are applied to the secondary database, which then waits for the next transaction to be applied on the system.

Snapshot replication is very similar to log shipping in that they both run in scheduled intervals and update the secondary database in mass change rather than applying each transaction to both systems as they are committed. Both technologies are utilized in much the same manner.

Replication requires specialized knowledge, which is a big concern for environments that don't have a dedicated DBA. Replication can be somewhat complex to troubleshoot and it does require a more involved design if it's to be used as a high-availability option.

Replication can appropriately fill the requirements for a high-availability solution. This technology does what database mirroring can do at a record level using transactional replication but without the option for an automated failover. Assuming you have sufficient resources and a bit of creativity, scripting a solution for an automated failover should not be out of reach.

Unlike database mirroring, both the source and the target databases are fully accessible to client applications. Replication provides the same functionality as log shipping with the use of snapshot replication.

Something to consider, however, is that the replication technology is battle-proven and well documented. Using replication for a high-availability solution has some drawbacks and performance can be a concern, but only as much as database mirroring is. Any high-availability solution you design with replication will most likely have a more complicated architecture to manage; not more advanced necessarily, but definitely more involved. In addition, one of the largest hurdles to consider is that if your database table structure changes or if you want to add a table to be replicated, you must break and redefine the publication for the changes to end up in both databases.

**Wrapping It All Up**

You can see now how creating a high availability solution for your environment requires a touch of creativity. Each SQL Server 2005 high-availability technology has its strengths and weaknesses and each lends itself to different situations.

Log shipping, snapshot replication and even database mirroring in high-performance mode would be good when geographically separating a primary Web database from the secondary database environments (particularly if they don't require that the secondary data is available in real-time.)

If, on the other hand, there is a secondary database requirement of real-time data, then transactional replication or database mirroring can carry the load if the transaction rates on the primary server are low and the link between the two environment sites is fast and not saturated.

Also consider your comfort level with these technologies. If you have working experience in some of the technologies already, you'll probably be fine. If you are a manager with no dedicated DBA, try to steer clear of the more complex technologies like replication where there are a lot of moving parts and troubleshooting can be complex. You might also consider contracting an experienced SQL Server consultant to help you design, implement and possibly train your staff to manage a new high availability suitable for your environment.

If high-availability in your organization requires only that the data is available a high percentage of the time and data downtime is taken very seriously, then clustering might be the option for you.

The point of all this is that SQL Server 2005 has provided a number of new options for implementing high availability that are tailored to serve different types of environments. A single availability option might fit your needs, or you may choose to take advantage of a combination of technologies, but as you now know, there are a variety of choices that are available to you.

**Zach Nichter** is a SQL Server professional with over 10 years of experience. He has held a number SQL Server support roles including DBA, team lead, manager, and consultant. Currently Zach is employed by Levi Strauss & Co. as the DBA Architect, focusing on SQL Server performance, monitoring, architecture, and other strategic initiatives.

© 2008 Microsoft Corporation and CMP Media, LLC. All rights reserved; reproduction in part or in whole without permission is prohibited.