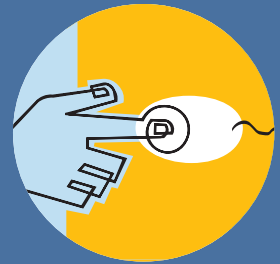
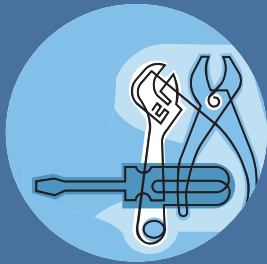


Real World SQL Server™ Disaster Recovery

A Survival Toolkit for the DBA



CLICK TO OPEN

By Brian Knight

SQL Server MVP, MCDBA, MCSE

Co-Founder SQLServerCentral.com

LUMIGENT

ABOUT THE AUTHORS

Brian Knight

Brian Knight, SQL Server MVP, MCDBA is a database manager at Fidelity National Financial and a co-founder of SQLServerCentral.com. He has worked with SQL Server for 10 years and has written several SQL Server books. Brian lives in Jacksonville, Florida with his better half and two children. He can be reached at bknight@sqlservercentral.com or by going to <http://www.sqlservercentral.com>.

TOOLKIT SCRIPT DEVELOPERS

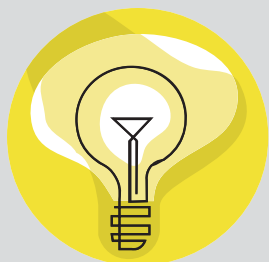
When I first saw the script that these two DBAs wrote during a DR drill, I said, "WOW!" If you haven't looked at it yet, take a look inside the package you received in this download.

Bruce Canaday

Bruce Canaday is a Database Consultant at Fidelity Information Services, a Fortune 500 provider of products and outsourced services and solutions to financial institutions and the real estate industry. He has more than 15 years of software development experience, specializing in relational database design and programming and has worked with SQL Server since version 4.2. He holds degrees in Industrial and Systems Engineering from the University of Florida and Computer Sciences from the University of North Florida. A native of Florida, Bruce currently lives in St. Augustine with his wife and three children.

Chris Gallelli

Chris Gallelli is a database administrator working for Fidelity National Finance. Chris is an MCP with more than 10 years of experience with SQL Server and was a contributing author to Microsoft SQL Server 2000 Unleashed. He has a Bachelor's degree in electrical engineering and a Master's degree in Business Administration from Union College. Chris currently lives near Albany, New York and can be reached at Cgallelli@excite.com.



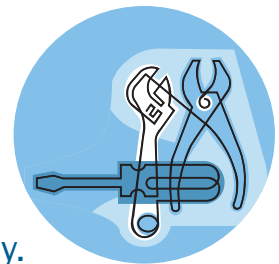
WHAT'S INSIDE YOUR TOOLKIT

	INTRODUCTION	4
CHAPTER 1	PLANNING FOR THE WORST	5
	Types of Disasters	6
	> Preparing your DR CD	7
	How Many Nine's are Acceptable?	7
	Planning For Failure	9
CHAPTER 2	TYPES OF BACKUPS	10
	Full	11
	> Performing a Full Backup in Enterprise Manager	11
	> Performing a Full Backup in T-SQL	14
	Differential	19
	Transaction Log	20
	File or Filegroup	23
	Typical Backup Scenarios	25
CHAPTER 3	TYPES OF RESTORES	27
	Full	28
	> Killing Users Quickly	33
	Differential	34
	Transaction Log	35
	> Recovering to a Mark	36
	Restoring the Master Database	38
	Restoring Databases to a New Server	39
	Rebuilding Other System Databases Quickly	42
CHAPTER 4	RECOVERY MODELS	43
	Simple	44
	Full	44
	Bulk-Logged	45
CHAPTER 5	THE DR TOOLKIT — RESTORING DATABASES FAST!	46
CHAPTER 6	SELECTIVE DATA RECOVERY	50
	Restoring Row-Level Data	51
	Restoring Dropped and Truncated Tables	52
	CONCLUSION	54



INTRODUCTION

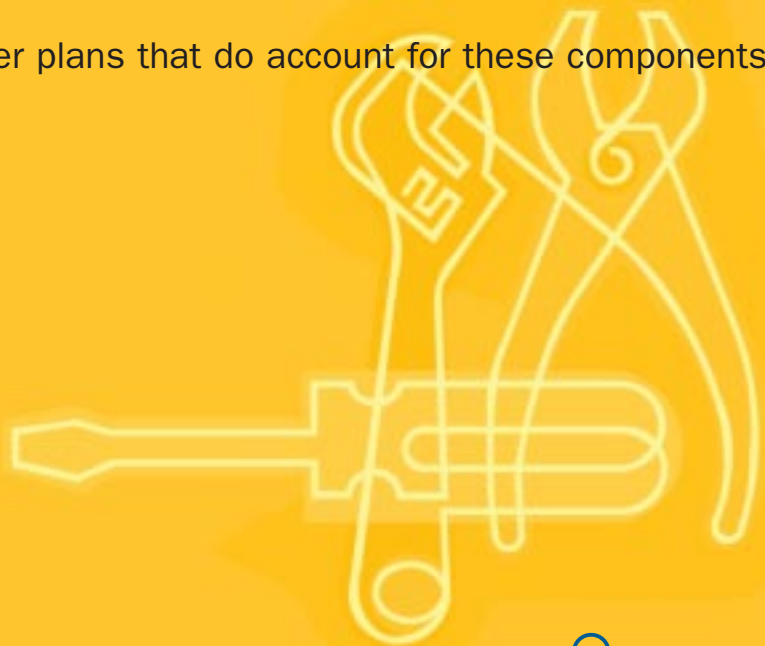
Disaster recovery(DR) planning and implementation are the trickiest tasks a DBA can do. It involves strong coordination between many groups that you may not professionally like and can come at a high financial and political cost. This DR Toolkit will walk you through planning for the worst and goes through many of the best practices in disaster recovery.



I also provide many tips that will help speed up your recovery using some very handy scripts that are provided as part of this download.

PLANNING FOR THE WORST

Proper planning, may easily account for 4 months of a 6 month deployment. Don't be discouraged though. You'll be a hero when (not if) you have to act on your plan. True, this toolkit is on database disaster recovery, but you must also consider other components in the environment. For example, if your network card in your database server malfunctions, can your database server remain online? How about a firewall? These components are far from a DBA's job to monitor or plan for but a database disaster recovery plan should at least have the contact information and links to the other plans that do account for these components.



TYPES OF DISASTERS

When you think of a database failure, any number of components can affect the availability of a database. Generally, it'll be components out of your control that may make your database unavailable. To truly prepare for a database failure though, you may have to go to other departments inside your company to confirm their plan has been validated and is well documented. For some large companies this could involve you getting on tights and a Superman costume to do this since you may have political red tape to get through. Some of the types of disasters that may affect the availability of your database can be seen in Table 1.

TABLE 1 DATABASE COMPONENT WORKSHEET

Type of Failure	Some of the Components	Plan Developed	Documented	Plan Validated
Site	Fire, bomb or hurricane shuts down your entire site	⊙	⊙	⊙
Network	Network card, router, connectivity	⊙	⊙	⊙
Peripheral Services	Firewall, domain controller	⊙	⊙	⊙
Storage	RAID controller, SAN, hard drive	⊙	⊙	⊙
Server	Operating system	⊙	⊙	⊙
Database	Database corruption, user error	⊙	⊙	⊙
Data	Operator error (user drops a table)	⊙	⊙	⊙
Process	Jobs, extracts, anything that feeds the database	⊙	⊙	⊙

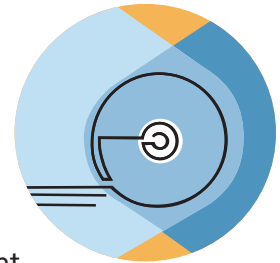
There are of course many other types of failures but this should at least give you a start. The best tip is to get all the groups in a single room and walk through each of the components and how they would handle a failure. Keep in mind also that a plan doesn't do any good unless it's well documented. Even though it's horrific to think, if you have a site disaster, you may not be around to actually execute the plan.

Preparing your DR CD

In the event of a real DR, you won't have time to find your SQL Server CDs and your service packs. Instead, it's better to have a set of CDs already prepared with everything you need with the exception of the actual backups to restore your SQL Server.

Some of the things you may need would include:

- ⊗ All of your documentation
 - SQL Server install documentation
 - Product installation
 - Disaster recovery plans
 - Any other documentation that may be needed to recover your environment
- ⊗ SQL Server install
- ⊗ The latest service pack and hot fixes
- ⊗ Any DTS packages that were saved as COM-structured file



HOW MANY NINE'S ARE ACCEPTABLE?

So, what are these nine's of availability that everyone is talking about? Essentially, it's the percentage your system is available.

For example, if you hear your manager ask for "five-nines" of availability, your manager is asking for 99.999% uptime on your systems (note that there are five nines in the percentage).

Calculating your availability is a simple formula that uses two major factors, which are generally measured in hours: Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR).

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} = .9xxxxx$$

While asking for 3-nines of availability seems like an easy task to achieve, it's actually a lot tougher than it sounds. For example, if you want to achieve only 3-nines of availability, you have less than 44 minutes a month that the system can be offline.

TABLE 2 NINES OF AVAILABILITY

Availability	Downtime Per Month	Downtime Per Year
99.9999%	2.6 seconds	32 seconds
99.999%	26.25 seconds	5 minutes, 15 seconds
99.99%	4 minutes, 23 seconds	52 minutes, 36 seconds
99.95%	21 minutes, 54 seconds	4 Hours, 23 minutes
99.9%	43 minutes, 50 seconds	8 Hours, 46 minutes
99.5%	3 hours, 39 minutes	1 day, 19 hours, 48 minutes
99%	7 hours, 30 minutes	3 days, 15 hours, 40 minutes

Keep in mind that the larger the database, the longer it is going to take to restore. It's not unheard of to take an entire day to restore a large database and then another day to perform the constancy checks. Even with high-end clustering solutions, three nines may be aggressive. Consider that NASDAQ has reached a highly respectable 99.97% level of availability with 200 transactions a second.

**TIP**

It's become quite popular now for application service providers (ASPs) to tie service level agreements (SLAs) to a given number of nines since it's a very measurable level of service quality. In these cases, missing the availability mark in the SLA contract will have financial penalties. Make sure when these are being negotiated that ample maintenance windows are built into the contract for reboots and applying fixes. These types of maintenance windows downtimes should not be included in the availability calculation.

**TIP**

While Windows clustering can solve the availability piece of the puzzle, it cannot solve the data loss problem. If your business owner tells you that no more than 15 minutes of data loss is acceptable, backups alone will not be the answer to your architecture. You'll need to stand up a hot site with a duplicate server ready to accept application requests. Typically, this site is kept up to date with log shipping, but if your executive wants less than 15 minutes of data loss, then you're more than likely looking at replication. Both of those solutions are out of the scope of this whitepaper since they're such a large topic by themselves.

PLANNING FOR FAILURE

Before you begin to come up with a strategy for disaster recovery, you must first know the requirements. Many of the requirements are financial, like your company will lose \$1.4 million for each hour your system is down. Many of the requirements may be political, like my largest client is on edge and my company will lose their account if we have another outage.

So, really an outage can be translated into two components:

- 1 How long can your system be down?
- 2 How much data loss is acceptable?

We've already talked about the first question, which is around availability. The second question, though, is a bit tougher to put your finger on. It's a question that can't be answered by you.

Instead, it must be answered by interviewing the business owners or executives. You must ask them questions like:

- 1 How much data loss is acceptable?
 - a. Just recreate the database from a template database
 - b. A week is fine
 - c. No more than a day
 - d. No more than an hour
 - e. No more than 5 minutes
 - f. None
- 2 If I were to lose my database in the middle of the day, what would be the cost to recreate the work?
- 3 If I were to lose my database in the middle of the day, what would be the hidden cost, such as perception to my clients?
- 4 If I were to lose my database in the middle of the day, would I lose my job?

Of course, the knee-jerk reaction to Question 1 when you interview an executive will almost always be "None". That's why the other questions try to flush out the real answer. Keep in mind also that as you go down the scale on acceptable data loss, it becomes much more expensive.

TYPES OF BACKUPS

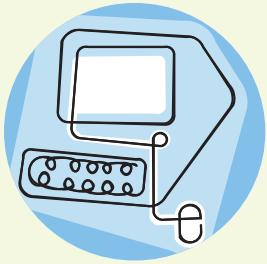
In this section, we'll explore the many types of backups you can use in SQL Server. I'll show you how to use both the Enterprise Manager and T-SQL. I always prefer using T-SQL to script my backups since you can reproduce the process by just changing a few keywords. Also, T-SQL is preferred because you'll have a few more options available to you than in the GUI.

NOTE: You must be a member of the sysadmin, db_owner, or db_backupoperator role to create a backup of a database. Additionally, you may need the password for the backup media.



FULL

A full backup is one of the simplest because it creates a stand-alone file that contains all the data, schema, indexes, users and permissions. While it gives you a great snapshot of your database at a given time, it is not meant for point-in-time recovery. This is because it does not compress the database especially well to perform every 15 minutes, as you would need in a real-time recovery. Typically you see full backups being made once a day or even less with large databases.



UNDER THE COVERS

So what happens if you're creating a backup right in the middle of a large transaction? SQL Server ensures that if the transaction began before the backup was command issued, it will be rolled into the backup file. Here are the steps that are performed when you issue a backup:

- 1 The database backup begins and the start time is noted.
- 2 Data pages are backed up in a sequential manner from each data file.
- 3 Transactions that have completed since step 1 are appended to the backup.

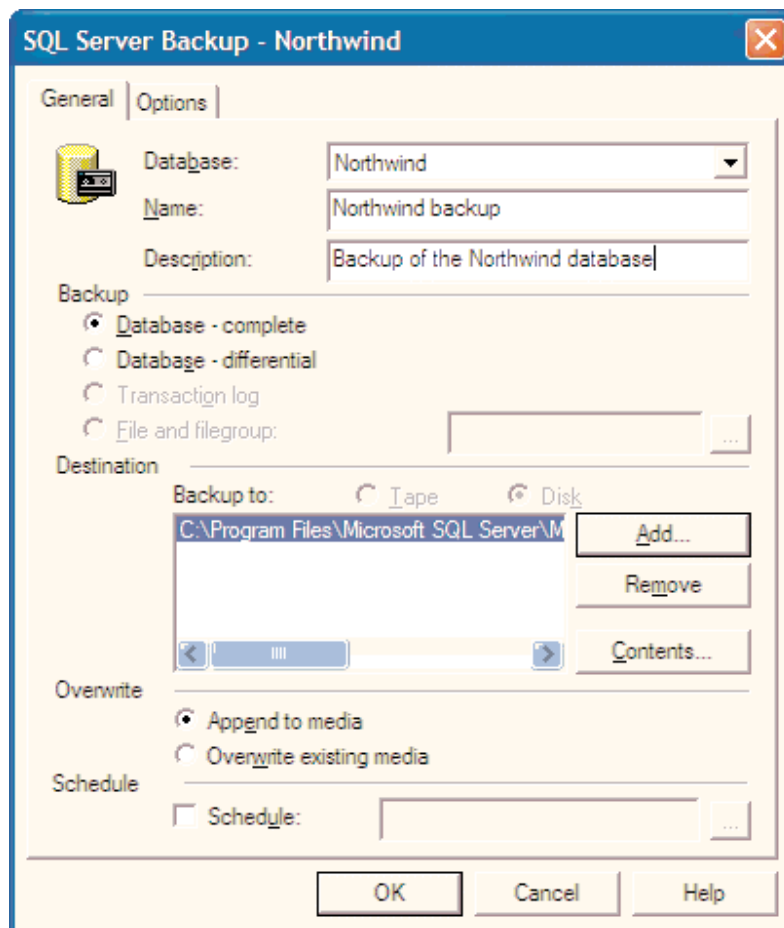
A full backup will back up your users, but not your logins—that is, unless you back up the master database. Once you restore your database to a different server, your users and your login IDs will be out of synch. I'll show you how to fix this later in this whitepaper.

Performing a Full Backup in Enterprise Manager

To back up a database in Enterprise Manager, right-click on the database and select **All Tasks | Backup Database**. In the SQL Server Backup screen (shown in Figure 1), most of the important settings are automatically set and are self-explanatory. The options that are available in this screen vary according to the recovery model your database is in, which we'll talk about later.

Select the “Database—complete” option to perform a full backup. If you have no tape device, only the disk option is available. Then, click Add if there’s not already a file or device in the destination window. In the Overwrite section, you can select Append to Media to write multiple backups to the same backup file. You can later select which one you’d like to restore. If you click Overwrite Existing Media, SQL Server will delete any backups previously made to the backup file and overwrite the file. You can also schedule the backup to occur by clicking the Schedule checkbox.

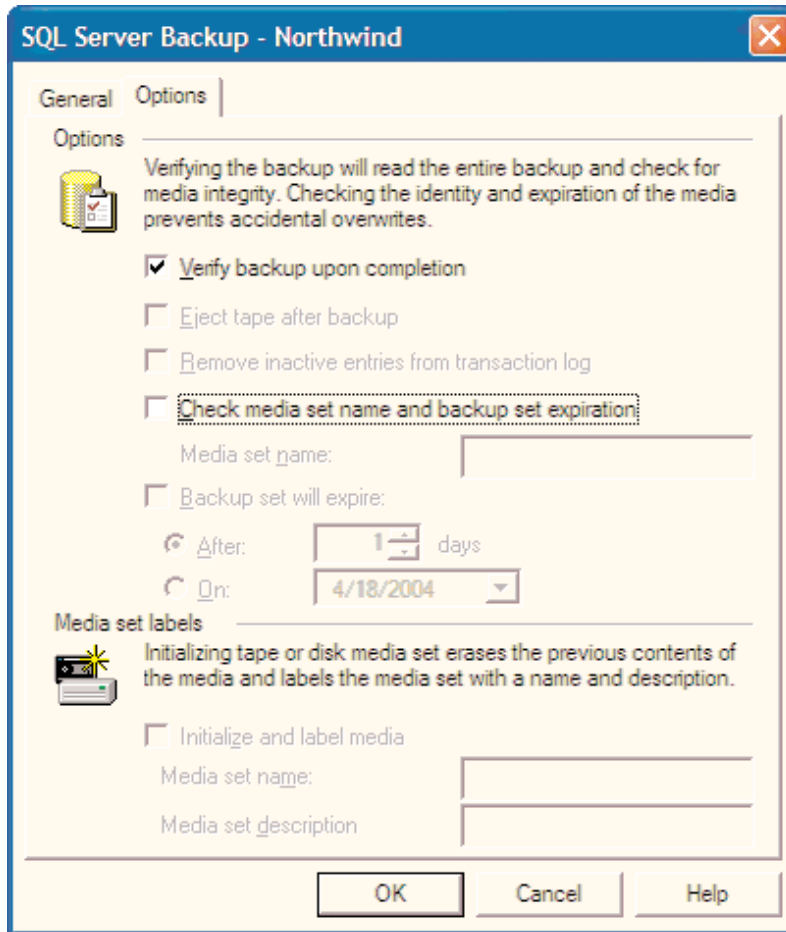
FIGURE 1 SQL SERVER BACKUP SCREEN



TIP

You can change the default directory for backups modifying the Backup Directory data item in the **HKEY_LOCAL_MACHINE\Software\Microsoft\MSSQLServer\MSSQLServer** registry key. This key may vary if you have a named instance.

Generally speaking, that’s all you’ll have to do to create the backup in Enterprise Manager. To complete the backup, just click OK. You do have additional options in the Options tab you can set if you have more demanding needs.

FIGURE 2 OPTIONS TAB IN THE BACKUP DIALOG BOX

At a glance, the options you have are:

- ⊗ **Verify Backup Upon Completion** This option compares the backup against the live database. If you check this option, your server's performance will drop while this operation occurs, but you increase the likelihood of catching a bad backup before it becomes a problem. I have never found a reason to not check this option.
- ⊗ **Eject Tape After Backup** This option ejects the media after it completes the backup. This could be a bad option to check if you're storing multiple iterations of the backup on one tape. It's handy if you want to be visually alerted when the backup completes.

- ⊗ *Remove Inactive Entries From Transaction Log* This option is only available when you're performing a transaction log backup. This option truncates the transaction log whenever the backup is complete. It is checked by default. If you don't want your transaction log backup files to be cumulative, deselect this option. There's generally not a reason to uncheck this option and if you do uncheck it, your transaction log could grow to be an enormous size quickly.
- ⊗ *Check Media Set Name and Backup Set Expiration* This option confirms the media name and expiration date of the media before overwriting the file. If you select this option, specify the name of the media set to check or leave it blank if you didn't specify a media set name.
- ⊗ *Backup Set Will Expire* Under this option, specify how long the backup will be kept before it can be overwritten. This option allows you to override the default retention time that is set in the main Server Properties dialog box in the Database Settings tab.
- ⊗ *Initialize and Label Media* This option deletes the contents of any existing backups in the backup file or device. You can also use the Media Set Name and Description options to help you locate your backups. This option is not available unless you selected the Overwrite button in the General tab.

Performing a Full Backup in T-SQL

Performing a backup in T-SQL is of course inherently trickier, but it provides some nice options that can give you more feedback on the status of the backup and a few more options. First of all, the syntax of performing a backup using T-SQL can be a bit long:

```

BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [DISK=<backup file>][ ,...n ]
[ WITH
  [ BLOCKSIZE = { <block size> | @blocksize_variable } ]
  [ [ , ] DESCRIPTION = { <'text'> | @text_variable } ]
  [ [ , ] DIFFERENTIAL ]
  [ [ , ] EXPIREDATE = { <date> | @date_var }
    | RETAIN_DAYS = { <days to retain> | @days_var } ]
  [ [ , ] PASSWORD = { <password> | @password_variable } ]
  [ [ , ] FORMAT | NOFORMAT ]
  [ [ , ] { INIT | NOINIT } ]
  [ [ , ] MEDIADESCRIPTION = { <'text'> | @text_variable } ]
  [ [ , ] MEDIANAME = { <media name> | @media_name_variable } ]
  [ [ , ] MEDIAPASSWORD = { <media password> | @mediapassword_variable } ]
  [ [ , ] NAME = { <backup set name> | @backup_set_name_var } ]
  [ [ , ] { NOSKIP | SKIP } ]
  [ [ , ] { NOREWIND | REWIND } ]
  [ [ , ] { NOUNLOAD | UNLOAD } ]
  [ [ , ] RESTART ]
  [ [ , ] STATS [ = <percentage> ] ]

```

It's not as bad as it looks though. To do a vanilla backup of the Northwind database, the syntax would look something like this:

```

BACKUP DATABASE Northwind
TO DISK = 'C:\MSSQL\northwind.bak'

```

This command returns the following results:

```

Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 5.
BACKUP DATABASE successfully processed 353 pages in 2.757 seconds (1.046 MB/sec).

```

If you ran the command multiple times, it would continuously append the backups to the same file and the backup file would grow until it was formatted or you issued the REINIT command. What's nice about the output it gives you is it's much more informational than the Enterprise Manager GUI. It lets you know how long it took to restore and how many MB a second it averaged. This is useful if you're trying to benchmark various backup scenarios. Finally, you can also see in the results that this backup represents file 5 in the backup file. This means that there are 4 other backups that have been made previously to this file.

As you can see in the syntax, you have more options available, as follows:

- ⊗ **BLOCKSIZE** The number of bytes per block that SQL Server uses to back up the database. SQL Server automatically finds the best setting for this option and, usually, you should not adjust it. In rare cases, some media devices such as CD-ROM writers may require a change in this setting.
- ⊗ **DESCRIPTION** A description of the backup that appears in the backup file's header. This setting is useful for organizational purposes when you're trying to restore the database.
- ⊗ **EXPIREDATE** The date when the backup can be purged.
- ⊗ **RETAIN DAYS** Specifies how many days after the backup the file is protected from overwriting.
- ⊗ **PASSWORD** Sets a password on the backup set. This prevents anyone from restoring the backup unless they know the password. This password does not prevent you from overwriting the backup file.
- ⊗ **FORMAT / NOFORMAT** The FORMAT option erases all header information on the media before backing up the database. This is handy when you want to format the tape drive to begin a new set of backups. You should use this option only when you're sure you won't need any old information on the tape. If you use the FORMAT option, the INIT and SKIP options are assumed.

- ⊗ *INIT / NOINIT* The INIT option erases all information on the backup set. NOINIT will append the backup to the end of the existing file.
- ⊗ *MEDIADESCRIPTION* This option adds a media description to the media, such as a tape drive. This allows a backup operator to quickly find the proper media since most third-party tools can read the description.
- ⊗ *MEDIANAME* This adds a media name to the media, such as a tape drive.
- ⊗ *NAME* This adds the name of the backup set and can contain up to 128 characters.
- ⊗ *NOSKIP / SKIP* The NOSKIP option tells SQL Server to check the media name and expiration date before overwriting the backup.
- ⊗ *NOREWIND / REWIND* The NOREWIND option is used to prevent other applications from using the tape until SQL Server issues a BACKUP or RESTORE command. If REWIND is specified, SQL Server releases control of the tape media and rewinds the tape.
- ⊗ *NOUNLOAD / UNLOAD* When UNLOAD is specified, the tape is rewound and unloaded.
- ⊗ *RESTART* On tape operations, this option restarts a backup operation that has failed previously.
- ⊗ *STATS* This option returns every specified percentage of backup completion. For example, a setting of 10 reports every 10 percent.

Let's now look at some slightly more complex syntax. You can achieve much greater performance in your backup of a large database by sending the backup to multiple files. While we want see a large difference with our tiny Northwind database, you may see huge gains with larger databases. This is especially useful if the backup files are on separate disks or, to a greater extent, separate disk controllers.

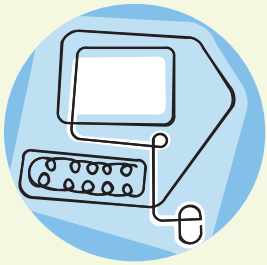
To backup to multiple files, you must only use the DISK option twice in your BACKUP command. Also in this example, we're going to raise the INIT option and FORMAT options to initialize and erase the old header information from the backup file. Keep in mind by doing this, we've removed all previous backups from this file. The last option we've included in this script is the STATS option, which will send the status to the client at every 20% milestone. This will hardly be noticeable in our Northwind backup but on a large database, the STATS option is a lifesaver to let you know that the end is in sight. So, the syntax for performing the mentioned options would look like this:

```
BACKUP DATABASE Northwind
TO DISK ='C:\MSSQL\northwind.bak',
    DISK ='D:\MSSQL\Northwind2.bak'
    WITH INIT , NAME = N'Northwind backup',
    STATS = 20, FORMAT
```

The results will look something like this and may vary based on your database:

```
21 percent backed up.
41 percent backed up.
60 percent backed up.
80 percent backed up.
Processed 328 pages for database 'Northwind', file 'Northwind' on file 1.
100 percent backed up.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP DATABASE successfully processed 329 pages in 1.173 seconds (2.291 MB/sec).
```

Notice that the percentage status is not exactly every 20%. This is because the database is so small.



UNDER THE COVERS

Once you create a backup file, a media set is created behind the scenes. Media sets allow SQL Server to tie multiple backup files together. If you were to go from the first backup script we covered in the first example to the second backup script in the last example without issuing the `FORMAT` command, the backup would fail with the following command:

```
Server: Msg 3259, Level 16, State 1, Line 1
The volume on device 'C:\Program Files\Microsoft SQL
Server\MSSQL\BACKUP\northwind.bak' is not part of a multiple family
media set. BACKUP WITH FORMAT can be used to form a new media set.
Server: Msg 3013, Level 16, State 1, Line 1
BACKUP DATABASE is terminating abnormally.
```

Luckily, the error is pretty explicit on what it wants you to fix. Once you issue the `FORMAT` command, it'll drop the old backups on the file and recreate the media set as a multiple-file media set. The same also holds true when you go from 4 files in a media set to 5 or 5 to 1.

DIFFERENTIAL

Differential backups are very similar to what many call incremental backups. In SQL Server terms, a differential backup will backup only the extents that have changed since the last full backup. This will make these types of backups substantially smaller than full backups and is really useful with large backups. For example, if you have a 500 GB database that is 24x7, full backups may take an hour to run. You may decide in this case to run a complete backup on Sundays then do differentials each night.

Note: In the above example, each day your differential backup would grow until the next complete backup ran.

To perform a differential backup in Enterprise Manager, simply select the "Database – differential" option that can be seen in Figure 1. In T-SQL, you can run the following syntax:

```
BACKUP DATABASE Northwind
TO DISK = N'C:\MSSQL\NorthwindDiff.bak'
WITH NOINIT , NOUNLOAD , DIFFERENTIAL , NAME = N'Northwind backup',
STATS = 10, NOFORMAT
```

This outputs results similar to those for a full backup. If the database doesn't have a lot of changes, SQL Server will increase the STAT parameter to compensate.

```
49 percent backed up.
99 percent backed up.
Processed 16 pages for database 'Northwind', file 'Northwind' on file 1.
100 percent backed up.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 17 pages in 0.472
seconds (0.279 MB/sec).
```

TRANSACTION LOG

Transaction log backups take a backup of whatever is in the transaction log. The transaction log can contain every INSERT, UPDATE, and DELETE statement made against the database. It may also contain data around other types of actions like:

- ⊗ Index creation or rebuilds
- ⊗ Bulk operations that occur in the database like BULK INSERT
- ⊗ Other DDL statements

This can vary based on the recovery model you have selected (more on this in the Recovery Model section). This gives you great flexibility with your recovery since transaction log backups allow you to restore to a given point-in-time or transaction.

As you can imagine, because almost every operation is logged, it can create a rather large transaction log quickly.

This is especially true during ETL processes or index rebuilds.

**TIP**

To determine how much space is being used by your transaction log, you can run `DBCC SQLPERF`. This command will tell you how much space is being used by your transaction log and how much space is available in the transaction log. If you have a transaction log that is only 10% full, you may want to shrink the transaction log. Keep in mind though there's no point in shrinking the log file if it's just going to increase to the same size that night. To issue the command, run the following syntax:

```
DBCC SQLPERF (LOGSPACE)
GO
```

And the results would look something like this (of course varying based on the databases you have).

Database Name	Log Size (MB)	Log Space Used (%)	Status
pubs	1.99219	4.26471	0
msdb	3.99219	17.0132	0
tempdb	1.99219	1.64216	0
model	1.0	12.7953	0
master	3.99219	14.3469	0

The transaction log will continue to fill and expand until you issue a `BACKUP LOG` statement (whether through the Enterprise Manager GUI or through T-SQL). So if you don't perform a transaction log backup often, your drive can fill up quite fast.

To backup the transaction log, simply select the Transaction Log option in Figure 1. This option may be grayed out if you have the Simple recovery model selected. We'll go more into the recovery models later in this whitepaper. There are a few added options in T-SQL for backing up transaction logs. Those are:

- ⊗ **`NO_LOG / TRUNCATE_ONLY`** These two synonymous options truncate the transaction log without backing it up. If you run this, you can no longer do point-in-time recovery.
- ⊗ **`NO_TRUNCATE`** This option allows you to back up the transaction log and not truncate it afterwards. This is handy when you're trying to back up a transaction log that may have a damaged database due to a physical device failure. Never let your log grow for a long time by specifying

To back up the transaction log in the Northwind database, use the following syntax:

```
BACKUP LOG Northwind TO
DISK = N'C:\MSSQL\NwindTran.bak'
WITH NOINIT , NOUNLOAD , NAME = N'Northwind backup',
STATS = 10, NOFORMAT
```

This outputs the following results:

```
80 percent backed up.
100 percent backed up.
Processed 10 pages for database 'Northwind', file 'Northwind_log' on file 5.
BACKUP LOG successfully processed 10 pages in 0.089 seconds (0.920 MB/sec).
```

If there is no data in the transaction log to be backed up, you receive the following message:

```
There is no current database backup. This log backup cannot be used to roll forward a preceding
database backup.
100 percent backed up.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 3.
BACKUP LOG successfully processed 1 pages in 0.137 seconds (0.007 MB/sec).
```

Note: If there is nothing to backup, which is quite regular in the Northwind database, your database maintenance plans may return an error. If you receive this error, you may want to rethink your backup strategy to just do full backups.

If the database is in Simple recovery model, then you will receive the following option. To fix this, change the database recovery model to Full or Bulk-logged. We'll discuss this a little later.

```
Server: Msg 4208, Level 16, State 1, Line 1
The statement BACKUP LOG is not allowed while the recovery model is SIMPLE. Use BACKUP
DATABASE or change the recovery model using ALTER DATABASE.
Server: Msg 3013, Level 16, State 1, Line 1
BACKUP LOG is terminating abnormally.
```

If you only want to truncate the transaction log, use the following syntax:

`BACKUP LOG Northwind
WITH NO_LOG`

FILE OR FILEGROUP

File or filegroup backups are a handy tool in the DBA toolbox to help backup large databases. As the name implies, a file or file group backup will backup a single file group or file in the database. It requires careful planning to make sure that you use the full benefit of the feature. It's used in conjunction with transaction log backups to ensure data integrity. For example, if you restore a file that contains the Employees table, the transaction log backups will recover to the point-in-time when the file backup was created. This ensures that no foreign key constraints are broken and that no data is left in an inconsistent state.

Because of the complexity of implementing these, I shy away from this backup type unless there's an absolute requirement for it or the database is too large to backup in a single window. The complexity revolves around distributing the data across multiple files to get the most benefit. I do this on larger databases but I see little benefit of doing this with the smaller SQL Server database.

The main benefit of file backups is that it provides protection from an isolated media failure. For example, if a single drive fails that contains a primary data file, you could restore it individually. To backup a filegroup in T-SQL, simply issue the following command (I've added some extra parameters that we have discussed earlier, but they're not required):

```
BACKUP DATABASE Northwind FILEGROUP = N'PRIMARY'  
TO DISK = N'C:\MSSQL\Northwind.bak'  
WITH NOINIT, NAME = N'Primary FileGroup Backup', NOSKIP, STATS = 10
```

This would return results similar to the following:

```
12 percent backed up.  
21 percent backed up.  
31 percent backed up.  
41 percent backed up.  
51 percent backed up.  
60 percent backed up.  
70 percent backed up.  
80 percent backed up.  
90 percent backed up.  
100 percent backed up.  
Processed 328 pages for database 'Northwind', file 'Northwind' on file 1.  
BACKUP DATABASE...FILE=<name> successfully processed 328 pages in 1.251  
seconds (2.147 MB/sec).
```

Keep in mind that this must be used in conjunction with transaction log backups. Because of this, you cannot have the database in Simple recovery model. If you do have it in this model, you'll receive the following error:

```
Server: Msg 4208, Level 16, State 1, Line 1  
The statement BACKUP DATABASE...FILE=<name> is not allowed while the recovery model is  
SIMPLE. Use BACKUP DATABASE or change the recovery model using ALTER DATABASE.  
Server: Msg 3013, Level 16, State 1, Line 1  
BACKUP DATABASE is terminating abnormally.
```

TYPICAL BACKUP SCENARIOS

Let's take a few example scenarios with databases of varying sizes. Our first database is a very typical SQL Server database. It is 80 MB in size and is not especially mission critical. The database cannot be down for more than 10 minutes and 24 hours of data loss is fine since it only holds application preference information for the client. The application is only used between 9 AM-5 PM.

First, go back and ask the business owners the same questions we discussed earlier:

- 1 How long can your system be down?
- 2 How much data loss is acceptable?

Based on one of the requirements that we mentioned in the first part of this section, we know the application cannot be down for more than 10 minutes. Because of this, you'll have to use SQL Server clustering. The client, though, is being very generous on the data-loss front. This allows us to be much less aggressive with our backup strategy. In this scenario the following backup schedule would probably suffice:

Day	Time	Type of backup
Monday – Sunday	2 AM	Full backup

Another typical scenario is where you have a 500 GB database that's mission critical. Your company can only lose no more than 15 minutes of data at the most. This database is quite active since it's an e-commerce database. The database is operational 24 hours a day, 7 days a week. Your slowest day of operation is Sunday.

In this scenario, you would only have time to perform a full backup once a week. Differential backups can be performed daily instead of full to reduce the backup time. Transaction log backups would need to occur every 15 minutes to ensure you meet your data loss requirements.

Day	Time	Type of backup
Sunday	2 AM	Full backup
Monday-Sunday	Every 15 minutes	Transaction log backup
Monday-Saturday	2 AM	Differential backup



TIP

Backup and restore speed can be increased significantly on large databases if you break the backups into several files and place the files on separate drives. RAID 1 drives will perform faster for backups than RAID 5. You also will experience much faster write performance to disk than directly to tape.

TYPES OF RESTORES

For every type of backup, there is a like restore method.

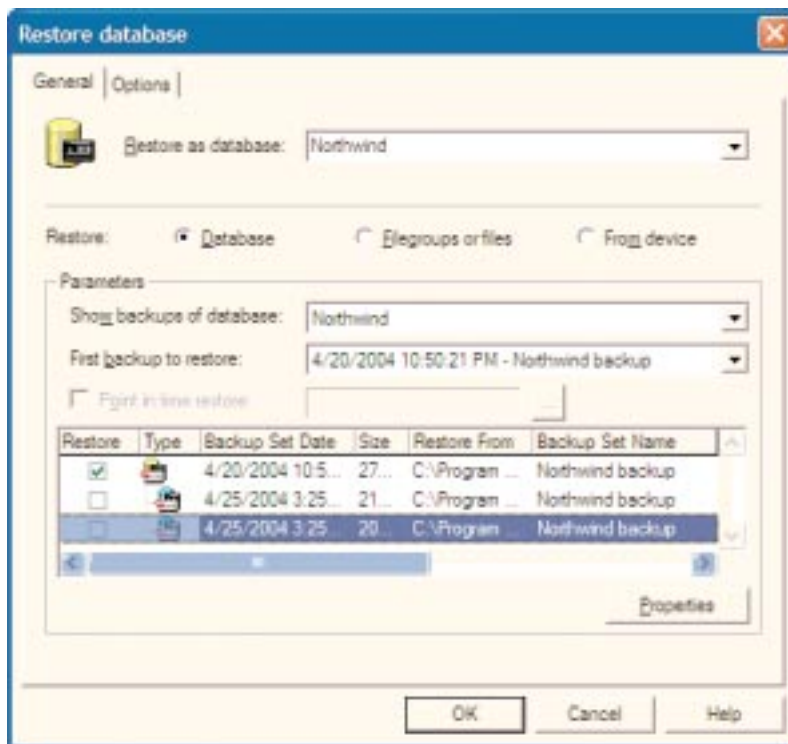
This section covers how you can restore your database to any given point in time in both Enterprise Manager or in T-SQL.



FULL

The first step in any database restore is to restore to a given base. You do this through a full database restore. To restore a database in Enterprise Manager, right-click on the database and select All Tasks | Restore Database. This opens the Restore Database screen seen in Figure 3.

FIGURE 3 PERFORMING A FULL RESTORE IN ENTERPRISE MANAGER



Once you pull up the Restore Database dialog box, you'll see the most recent database backups chronologically. Under the Restore As Database option, you can specify the database you'd like to restore or create. You can either select an existing database to restore to or type in the name of a nonexistent database to create and load. The easiest method of restoring your database is to select the Database option from the Restore radio buttons.

For the Show Backups Of Database option, select the database for which you created the backups. You can also select another database if you want to quickly migrate data from one database

to another. Once you select the database and the First Backup To Restore option, you need to check the backups to restore from the Restore column. If you've performed transaction log backups, the Point In Time Restore option is also available. In the Options tab, you have the following options available:

- ⊗ *Eject Tapes (if any) After Restoring Each Backup* Specifies that the backup media be ejected after each restoring the database by selecting the Eject Tapes After Restoring Each Backup option. If you don't have tape media installed on the server, this option is ignored.
- ⊗ *Prompt Before Restoring Each Backup* Presents a dialog box as SQL Server completes each backup and moves on to the next. This is handy when you're restoring transaction log backups and need to be alerted when SQL Server reaches each backup file.
- ⊗ *Force Restore Over Existing Database* Overwrites the existing database and log files. If you try to overwrite a database without checking this option, you receive an error.
- ⊗ *Restore Database Files As* This is the equivalent of the MOVE option. This allows you to change the path or filename you're restoring to. The files do not have to exist to restore to them. If they don't exist, SQL Server creates them automatically. If, for example, you're restoring to the Northwind database and want the database files moved to a different drive, you can specify the new drive and filenames here. After the restore, the old files are deleted.
- ⊗ *Leave Database Operational No Additional Transaction Logs Can Be Restored* Closes the restore and does not allow you to restore any more transaction log backups.
- ⊗ *Leave Database Nonoperational But Able To Restore Additional Transaction Logs* Leaves the database in the state of Loading until a restore is performed with the Leave Database Operational option selected.
- ⊗ *Leave Database Read-Only And Able To Restore Additional Transaction Logs* Leaves the database in read-only mode

until additional restores are made and a final restore made with the Leave Database Operational option selected.

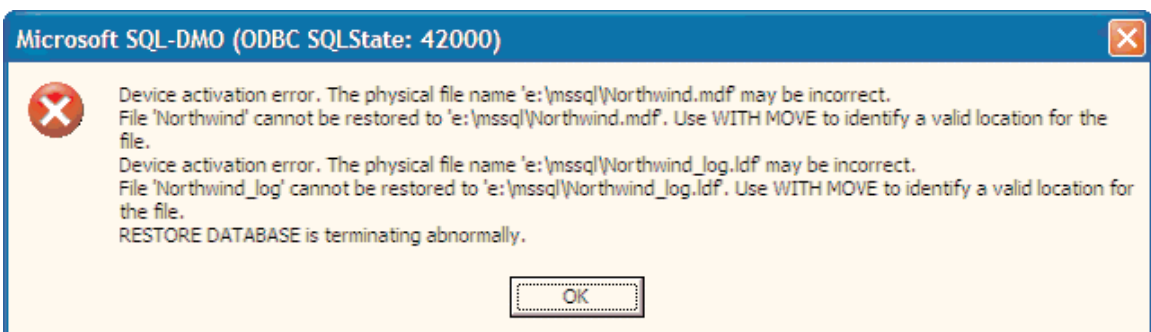
⊗ *Undo File* Specifies the path and filename of the undo file.

After you have all of your options selected, you can click OK to restore the database. Often, you'll have the need to restore a database from a different system or from a file that is not on the list of recent backups. You can do this by selecting the From Device radio button in the General tab of the Restore Database screen. Once you select the option, click the Add button to add the devices or backup files to restore.

If you restore a database from a different server you may experience the “WITH MOVE” error (shown in Figure 4), which looks pretty frightening at first sight. This error will only appear if the database has not been created and you're restoring a database from a different server that didn't match the same directory path structure for the data and log file.

For example, if you take a database from a production server where the data and log files are stored in a E:\MSSQL drive and folder, but the target server didn't have that drive or folder, you would see this error. This is because SQL Server is reading the backup file to determine where to create the data and transaction log files versus from the already created database. To fix the issue, simply go to the Options tab and enter a valid directory and drive letter and try to restore again.

FIGURE 4 "WITH MOVE" ERROR YOU MAY EXPERIENCE



In T-SQL the you can restore a database by using the following syntax:

```
RESTORE DATABASE { <database name> }
[ FROM < backup device | DISK <filename>> [ ,...n ] ]
[ WITH
  [ RESTRICTED_USER ]
  [ [ , ] FILE = { <file number> } ]
  [ [ , ] PASSWORD = { <password> } ]
  [ [ , ] MEDIANAME = { <media name> } ]
  [ [ , ] MEDIAPASSWORD = { <mediapassword> } ]
  [ [ , ] MOVE 'logical file name' TO 'OS path and file name' ]
    [ ,...n ]
  [ [ , ] KEEP_REPLICATION ]
  [ [ , ] { NORECOVERY | RECOVERY | STANDBY = undo_file_name } ]
  [ [ , ] { NOREWIND | REWIND } ]
  [ [ , ] { NOUNLOAD | UNLOAD } ]
  [ [ , ] REPLACE ]
  [ [ , ] RESTART ]
  [ [ , ] STATS [ = percentage ] ] ] ]
```

Some of the important options in this syntax are:

- ⊗ **RESTRICTED_USER** Restricts access to the newly restored database to only members of the db_owner, dbcreator, or sysadmin role.
- ⊗ **MOVE** Used to restore the database to a different path and filename. This is especially useful when you restore a database to a different server that may not have the same directory structure.
- ⊗ **NORECOVERY | RECOVERY** The NORECOVERY option is used to keep the database in a restored state to allow the administrator to apply more differential or transaction log backups. The RECOVERY option specifies that there are no further backups to apply. If you don't specify one of these options, the RECOVERY option is assumed.
- ⊗ **REPLACE** Creates the database and its files even if a database with the same name already exists.

- ⊗ *RESTART* Restarts the restore operation at the point of failure when restoring a database from a tape.
- ⊗ *KEEP_REPLICATION* Does not destroy your replication settings when restoring the database. Is not allowed in conjunction with the NORECOVERY option.

To restore the Northwind database from a dump file, you can use the following syntax:

```
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH RECOVERY, STATS = 25
```

Should result in:

```
26 percent restored.
51 percent restored.
75 percent restored.
100 percent restored.
Processed 328 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 329 pages in 0.853 seconds (3.151 MB/sec).
```

There are a few errors that you may receive when trying to restore the database backup. For example, if you try to restore a file backup that was created as a file or file group backup, you'll receive the following error:

```
Server: Msg 3135, Level 16, State 2, Line 1
The backup set in file 'C:\MSSQL\Northwind.bak' was created by BACKUP
DATABASE...FILE=<name> and cannot be used for this restore operation.
Server: Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
```

For that reason, it's a good idea to label the backup file based on the type of backup it is. For example, Northwind.TRN for a transaction log backup or Northwind.fil for a file backup. If you try to restore a single file from a mult-file backupset, you'll receive the following error:

```
Server: Msg 3132, Level 16, State 1, Line 1
The media set for database 'Northwind' has 2 family members but only 1 are provided. All
members must be provided.
Server: Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally
```

Again, simply naming the files appropriately will fix any confusion. For example naming the first file in a backupset Northwind1.bak. I've spoken earlier about the WITH MOVE error in Enterprise Manager. In T-SQL, you can get around the error by using the MOVE syntax for each of the data and log files as shown here:

```
RESTORE DATABASE NewNorthwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH MOVE 'Northwind' TO 'd:\newfiledb.mdf',
MOVE 'Northwind_log' TO 'd:\newfiledb.ldf'
```

KILLING USERS QUICKLY

As you may have already seen by following some of these examples in this whitepaper or in real life, restoring databases in product isn't as easy as I'm laying out here. SQL Server 7.0 and 2000 requires exclusive access to the database to restore the database. If you try to restore a database when another user is in it, you will receive an error similar to this:

```
Server: Msg 3101, Level 16, State 2, Line 1
Exclusive access could not be obtained because the database is in use.
Server: Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
Server: Msg 4306, Level 16, State 1, Line 1
The preceding restore operation did not specify WITH NORECOVERY or WITH STANDBY.
Restart the restore sequence, specifying WITH NORECOVERY or WITH STANDBY for all but
the final step.
Server: Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
```

The only way to fix this is to issue a KILL statement for each user in the database or wait for all the users to disconnect. The KILL statement will disconnect a user from a database. The problem with this is by the time you disconnect one user, five more may connect. You really need some sort of automated way to disconnect all the users in a database.

Included in this DR Toolkit is a script called `usp_killusers`. This will disconnect all the users in a specified database. To use it, just compile it into the master database and then execute `usp_killusers`, specifying the database name you'd like to disconnect the users from as shown below:

```
master..usp_killusers 'Northwind'
```

The output will show you the spids that are disconnected.

Killing Users

[Killing 57](#)

[Killing 58](#)

[Killing 59](#)

DIFFERENTIAL

Differential backup restores are performed by first restoring the latest complete full backup with the `NORECOVERY` option then applying the differential backup over it. Remember that the `NORECOVERY` option keeps the database in an open, non-operational mode until the next backup is applied with the `RECOVERY` option.

To perform this in Enterprise Manager, simply check the differential backup or restore the full in `NORECOVERY` mode and then overlay the differential backup. In T-SQL you can perform the same function using the `RESTORE` statement as shown below. In the next example, both the full and differential backups are on the same backup file:

```
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH FILE = 1, NORECOVERY
GO
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH FILE = 2, RECOVERY
GO
```

This would result in output like this:

```
Processed 280 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 281 pages in 0.551 seconds (4.167 MB/sec).
Processed 16 pages for database 'Northwind', file 'Northwind' on file 2.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 2.
RESTORE DATABASE successfully processed 17 pages in 0.356 seconds (0.371 MB/sec).
```

TRANSACTION LOG

Transaction log restores are done similarly to differential restores. First restore the full database backup then apply each of the transaction log files in the chronological order starting from the oldest backup to the newest. On the last backup, you can specify what time you'd like to restore to.

In Enterprise Manager, apply the backups the same way you apply differential backups. In T-SQL, you would restore them with a slightly different syntax of RESTORE LOG as shown below:

```
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH FILE = 1, NORECOVERY
GO
RESTORE LOG Northwind
FROM DISK = N'C:\MSSQL\NorthwindTran.trn'
WITH FILE = 1,
```

You can also use the STOPAT parameter to specify an exact date and time to restore the transaction log to:

```
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH FILE = 1, NORECOVERY
GO
RESTORE LOG Northwind
FROM DISK = N'C:\MSSQL\NorthwindTran.trn'
WITH FILE = 1, STOPAT = 'Mar 18, 2001 9:56 AM'
GO
```

RECOVERING TO A MARK

One of the hidden features in SQL Server 2000 is the concept of recovering to mark. This is quite useful if you have a large extract process that occurs each night at a random time.

Because of the random nature of the extract, you would gain a lot of benefit by "marking" the transaction that loads the data by using the WITH MARK option when creating the transaction. If the transaction loads bad data or an untrapped error occurs, you can restore back to before or after the transaction by using the STOPATMARK or STOPBEFOREMARK option.

The STOPATMARK option will restore to right after the marked transaction completed. If you use the STOPBEFORE option in the transaction log restore, SQL Server will restore the transaction log to right before the specified transaction began.



TIP

Marked transactions are also a useful way to roll back a change you made into production without having to guess the exact time you finished the change.

To mark a transaction, you must use the WITH MARK option in the BEGIN TRANSACTION command. To simulate this behavior, I began a transaction that was marked with the name NightlyETL. Then, I performed the risky transaction of deleting all my orders from the [Order Details] table.

```
BEGIN TRANSACTION NightlyETL WITH MARK
DELETE FROM [order details]
COMMIT TRANSACTION NightlyETL
```

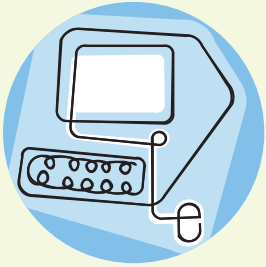
Once you have created the transaction log backups, you'll be able to issue the restore. In the below example, I have all the transaction log and full backups appending to the same backup file. The below script will allow me to restore to right before I deleted the [order details] table.

```
RESTORE DATABASE Northwind
FROM DISK = N'C:\MSSQL\Northwind.bak'
WITH FILE = 1, NORECOVERY
GO
RESTORE LOG Northwind
FROM DISK = N'C:\MSSQL\Northwind.BAK'
WITH FILE = 2, NORECOVERY
GO
RESTORE LOG Northwind
FROM DISK = N'C:\MSSQL\Northwind.BAK'
WITH FILE = 3, STOPBEFOREMARK = 'NightlyETL', RECOVERY
GO
```

The results of this would look similar to this:

```
Processed 280 pages for database 'Northwind', file 'Northwind' on file 1.
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 1.
RESTORE DATABASE successfully processed 281 pages in 0.513 seconds (4.475 MB/sec).
Processed 101 pages for database 'Northwind', file 'Northwind_log' on file 2.
RESTORE LOG successfully processed 101 pages in 0.261 seconds (3.156 MB/sec).
Processed 1 pages for database 'Northwind', file 'Northwind_log' on file 3.
RESTORE LOG successfully processed 1 pages in 0.088 seconds (0.005 MB/sec).
```

Note: Keep in mind that you can only recover to transaction marks if your database is in the full or bulk-logged recovery models.



UNDER THE COVERS

If you'd like to see what transaction marks have been made in your database and who has made them, you can select against the logmarkhistory table in the msdb database. The table holds results similar to this:

database_ name	mark_ name	description	user_ name	lsn	mark_ time
Northwind	NightlyETL	NULL	EDEN\ Brian Knight	23000000003600002	2004-05-09 13:17:11.290

RESTORING THE MASTER DATABASE

One of the most difficult situations to resolve in a DR scenario is a corrupt or bad master database. The master database contains the vital information SQL Server needs. It contains items like:

- ⊗ All of the logins and passwords for your server
- ⊗ A list of databases on your server with pointers to the files
- ⊗ The configuration of your SQL Server

If your master database is corrupt, your SQL Server will not start. To restore a master database in this situation, you can follow these steps to recover from the disaster:

- 1 Rebuild the master database by using the rebuildm.exe file located in the \Program Files\Microsoft SQL Server\80\Tools\Binn directory. Rebuilding the master database leaves your database files intact. It is always a good idea to back up the data and log files to a separate directory, just in case.
- 2 Restart SQL Server in single user mode by starting SQL Server with the -m parameter. This can be done in the Services applet under Control Panel | Administrative Tools.
- 3 Connect quickly after starting the service (someone else may grab your only connection if you don't). Restore your master database from the last known good backup.

- 4 Verify that the master database was successfully restored: confirm that all the databases are up and running. Restore the msdb database from the last known good backup and resize the tempdb database.
- 5 Stop and start SQL Server in normal mode by removing the `-m` switch from the startup parameters.
- 6 Open your database for production users.

If this does not work, rebuild the master database and then attach the databases. This task is also one of those DBA tasks that you hope you don't have to perform often. If the master database has problems, every database on the server has problems. Make sure you practice this on a test server often since it's a complex set of tasks if it's not practiced.



After you restore or attach a database to a different server, some of your users may not be able to connect to the database

RESTORING DATABASES TO A NEW SERVER

When you restore a database from a different server it's important to know the differences between logins and users. SQL Server has two basic levels of security: logins and users. Logins allow access to the server and hold server-level permissions. For example, if the user is a System Administrator (sysadmin) of your SQL Server, that is at a login-level. Users are at the database-level and hold permissions to individual objects. Also, users are assigned database roles such as `db_datareader`. Simply put, when you attempt to log in to the server, SQL Server validates your login, but as you attempt to access items in a database, SQL Server validates your user account.

After you restore or attach a database to a different server, some of your users may not be able to connect to the database. This is because the link between your SQL Server's logins and users almost always gets broken. When you add a login to a SQL Server, an entry is placed into your `syslogins` table in the master database. When the entry is made into this table, the login is also assigned a unique ID in the `SID` column (SUID in SQL Server 7.0). When you create a user in a database, it creates an entry in the `sysusers` table of the database.

This table contains a mapping back to the syslogin table with the SID, as shown here:

MASTER..SYSLOGINS DATABASE		NORTHWIND..SYSUSERS	
SID	Login Name	SID	User Name
1	SA	1	Dbo
10	Brian	10	Brian
11	Jenn	11	Jenn

For brevity, I've simplified the value of the SID.

Actually, a SID is a very long GUID, such as 0x1AD2D68CF6330E47865C9494DEB8918A. Once you restore a database from another server, the SID information will not match. As you can see in the following table, when you restore the database onto the separate server, your logins no longer will match your users. This may, in some rare cases, give permissions to the tables to users who don't need access and, more often, not allow in users who should have access. The SIDs that don't match are denied access to the database objects.

MASTER..SYSLOGINS DATABASE		NORTHWIND..SYSUSERS	
SID	Login Name	SID	User Name
1	SA	1	Dbo
10	John	10	Brian
11	Mary	11	Jenn
12	Brian		
13	Jenn		

A symptom of this problem appears when you see the user in the sysusers table, but not in Enterprise Manager under Users in the restored database.

The primary way to fix this login problem is to use the `sp_change_users_login` system stored procedure. This stored procedure rematches the users to the appropriate logins.

This procedure should be run only from the restored database after the logins have been recreated on the new system. To determine which logins don't match, you can run the stored procedure with the Report parameter as shown here:

```
sp_change_users_login 'report'
```

This outputs the following results:

UserName	UserSID
Brian	0xC7492F53646AD411934F0090273AC6
Kathy	0x5C2C14C4376ED411934F0090273AC6

To correct user names, use the Auto_Fix parameter as shown here:

```
sp_change_users_login 'autofix', Brian
```

If the login does not exist yet, this stored procedure will create them on the system. You have to fix the login's password and assign users back to any server roles they were placed in previously. This is why I prefer to have the logins already created on the second system before I run this stored procedure. After running the procedure, users have their original permissions re-established, as demonstrated here:

New login created.

Barring a conflict, the row for user 'Brian' will be fixed by updating its link to a new login.

Consider changing the new password from null.

The number of orphaned users fixed by updating users was 0.

The number of orphaned users fixed by adding new logins and then updating users was 1.

Another way you can use the `sp_change_users_login` stored procedure is to use the `Update_One` parameter. With this parameter, you explicitly designate which user on the restored database should map to which login on the server. You can use the following syntax to fix a user with this parameter:

```
sp_change_users_login 'Update_One', <user name>, <login name>
```

REBUILDING OTHER SYSTEM DATABASES QUICKLY

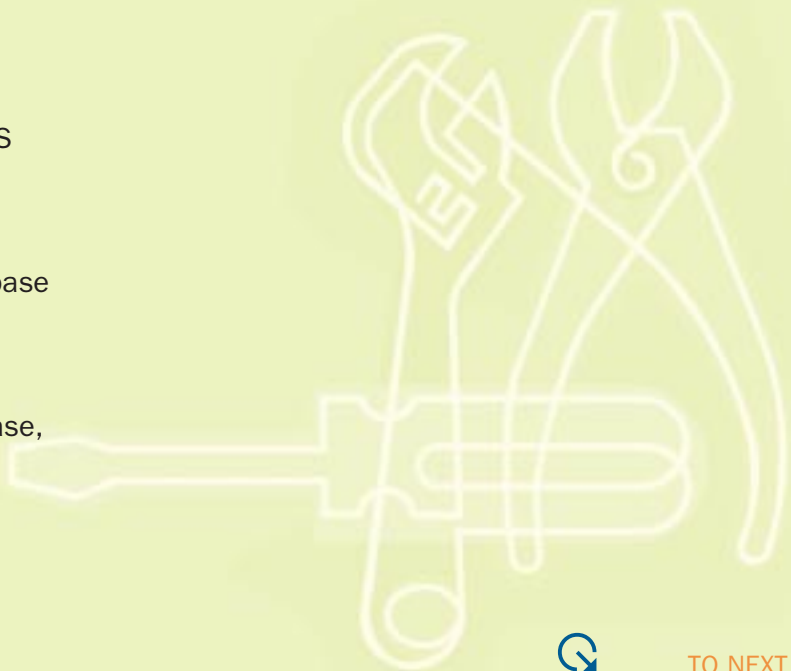
In some cases you can rebuild system databases without having to restore at all. For example, the Northwind database only contains sample data in it. Rather than worrying about backing up this sample database, you can just rebuild it in case you need it. In these cases, you can run the appropriate scripts from the `\Program Files\Microsoft SQL Server\MSSQL\Install` directory to rebuild these three databases. Use the following files to rebuild these databases:

- ⊗ **instmsdb.sql** msdb database
- ⊗ **instnwnd.sql** Northwind database
- ⊗ **instpubs.sql** pubs database

CAUTION

After rebuilding the msdb database, you will lose all DTS packages, jobs, operators, and other critical information.

Try to restore the msdb database from a known good backup first before rebuilding it. After you rebuild the master database, the msdb database is also rebuilt as a side effect.



RECOVERY MODELS

I have eluded several times to the different recovery models. SQL Server 2000 has a concept of recovery models, which lets you specify what your disaster recovery strategy is for your server at a database-level. There are three types of recovery models: simple, full, and bulk-logged. To set the recovery model in Enterprise Manager, simply right-click on the database you'd like to change and select Properties. You'll find the Recovery Model option in the Options tab.



You can also set the recovery model by using the ALTER DATABASE command as shown below:

```
ALTER DATABASE Northwind  
SET RECOVERY Full
```

When creating a new database, SQL Server will base its recovery model from the model database. So if you'd like all new databases on your server to have a certain recovery model, simply change the model database. The default recovery model for SQL Server Standard and Enterprise Editions is full and simple for Personal edition.

SIMPLE

The Simple Recovery model is the easiest recovery model to implement. This model is essentially the same as leaving the trunc. log on checkpoint option checked in SQL Server 7.0. The Simple Recovery model periodically truncates the transaction log, removing any transactions that have been committed. Because the transaction log is constantly being truncated, it cannot be backed up. This leaves you with only full and differential backups available for your backup strategy. This model is ideal for databases that don't change that often and are not mission critical. I typically leave development databases in this model.

FULL

The Full Recovery model provides the least chance of losing data, but you add administration and space costs. In this model, SQL Server logs all operations. This includes rows written through bulk operations like bcp or BULK INSERT. With the Full Recovery model, you can recover to any point in time as long as you're performing regular transaction log backups. Keep in mind that if you choose this option, your transaction logs and backups of the logs will grow fast in a rapid OLTP environment.

BULK-LOGGED

The Bulk-Logged Recovery model was designed as a compromise to the Full Recovery model. This model provides better performance and utilization of space compared to the Full Recovery model. This is because when a bulk operation occurs under a database with this recovery model enabled, SQL Server only logs the fact that the bulk operation occurred and which extent it occurred on. Since the bulk operation is not fully logged, your transaction logs remain much smaller than in the Full Recovery model.

Since the extents where the bulk operation occurred are logged, you can recover to a given point in time if your transaction log backups are performed regularly. The trade-off is that when you back up the transaction log, the extents where the data changed must also be backed up in addition to the transaction log. This means your transaction log backups can be quite large and they will take longer to finish.

THE DR TOOLKIT — RESTORING DATABASES FAST!

Now that you know how to do things the hard way, let's look at the DR Toolkit scripts, which will make your disaster recovery of a large database environment much easier. The DR Toolkit includes the following to aid in your recovery (in order of how you'd run these in production). You will want to compile the first two into your master database before proceeding.

- 1** `sp_CSS_RestoreDir`
- 2** `sp_Script_Logins`
- 3** Run `SyncAllDBsAllUsersScript.sql`



The first script to run is `sp_CSS_RestoreDir`. This script will restore all the backup files in a given directory. If you use the below syntax, the script will restore the backups to the specified location. The first parameter specifies what directory stores all of your backup files. The second parameter specifies where you'd like your data files to be restored to and the last parameter specifies where you'd like your transaction log files to be restored to.

```
exec sp_CSS_RestoreDir 'C:\sqlldb\sql_backup', 'C:\sqlldb\sql_data', 'C:\sqlldb\sql_log'
```

Alternatively, you can use the following syntax to restore the backups to their original location. The first parameter specifies where your backup files are located at and the `@MatchFileList` parameter tells the stored procedure to restore the files into their original drive and folder. Secondary data files (NDF) can also be restored into their own drive and folder with this parameter.

```
sp_CSS_RestoreDir 'C:\sqlldb\sql_backup', @MatchFileList = 'Y'
```

The stored procedure will name each database their original name and overwrite the databases that already exist with the same name. The results with either parameter will look like this:

filename

```
Northwind_db_200405101352.BAK
Northwind2_db_200405101352.BAK
Northwind3_db_200405101352.BAK
Northwind4_db_200405101352.BAK
Northwind5_db_200405101352.BAK
```

The big hitch with this stored procedure is that all the backups need to be in the same directory. Make sure the master database isn't one of the ones you restore this way since normal restoration practices don't apply for this database. Instead, I would restore the master database under a different name, like master2. This will allow you to run the next stored procedure with no parameters while connected to the new master2 database:

sp_Script_Logins

This stored procedure will output all the logins from the old master database and their respective server roles. It will also encrypt the password and create the appropriate password on the new server. To create the logins, passwords, and server roles on the new server, simply copy the output of sp_Script_Logins and paste it into a new query window. This output that you copy will look like this:

```
-- Login: bknight
SET @pwd = CONVERT (varbinary(256),
0x0100046D774224R73D7D7925BD26C4A19B863BC19849fj9DBD620F73543E1D90FFF69F9B
66C11BFE7DF014C7EFA4)
EXEC master..sp_addlogin 'bknight', @pwd, @encryptopt = 'skip_encryption'
if exists (select 1 from master..sysdatabases where name = 'master')
EXEC sp_defaultdb @loginame = 'bknight', @defdb = 'master'
EXEC sp_addsrvrolemember @loginame = 'bknight', @rolename = 'sysadmin'
```

Notice in this script that the output that you copy and paste into the new query window also will set the default database for the login. In reality, you'll probably see many more logins. System logins like sa are not outputted. If the login already exists on the new server it is not created and the password is not changed.

The last script will synch the logins with the users on the database if the backups originated from a different SQL Server instance. Remember, if you restore a database from a different SQL Server instance, the login will not be synched with the user and you'll have to run `sp_change_users_login` to correct the issue. This code will do this for every database on the server. Execute the contents of the `SyncAllDBsAllUsersScript.sql` script that was provided to you in the DR Toolkit and the output will correct any logins that are out of synch. You will then need to copy and paste the output of the stored procedure into a new query window and execute it. The output you copy and paste will look much like this:

```
USE [Northwind2]
GO
```

```
exec sp_change_users_login 'UPDATE_ONE','bknight','bknight'
exec sp_change_users_login 'UPDATE_ONE','TEST1','TEST1'
```

```
USE [Northwind3]
GO
```

```
exec sp_change_users_login 'UPDATE_ONE','bknight','bknight'
exec sp_change_users_login 'UPDATE_ONE','TEST1','TEST1'
```



SELECTIVE DATA RECOVERY

SQL Server 7.0 and 2000 does not have the capability to restore data in some cases. For example, you cannot restore data at a table-level. If you restore the transaction log, you cannot select what data you'd like to restore other than to a given time of day. If you'd like to restore data at table-level, you would have to restore the database onto a parallel database then copy the table over selectively onto the damaged system. This works well until your database is too large to have two copies. For example, it's generally hard to find a 500 GB drive system to support you restoring a production database.



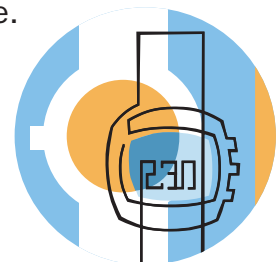
Never fear though, there are ways around this problem. A product called Lumigent® Log Explorer®, which offers online transaction analysis and data recovery, can get you out of some sticky disasters. Among its core functionality, it can restore data at a record or table level selectively. Best of all, the product can function while your database is active and online. It solves two major feature flaws with SQL Server restorations: being able to restore selective data and having users querying your database while you restore data. To install the product, go to <http://www.lumigent.com> and select downloads. The product has a client and server install. Both installs can be done from the same workstation. Once installed, this section will explain how you can use some of the features for disaster recovery of certain problems.

RESTORING ROW-LEVEL DATA

You can protect your system from a lot of things but the most dangerous thing to protect your server from is a dumb user who runs the wrong type of command. For example, if a user selects the wrong customer in the application to delete.

This type of action can normally be recovered by restoring the transaction log to a point in time, but you experience problems with this plan if the user doesn't tell you about the problem in 20 minutes. Rather than lose the

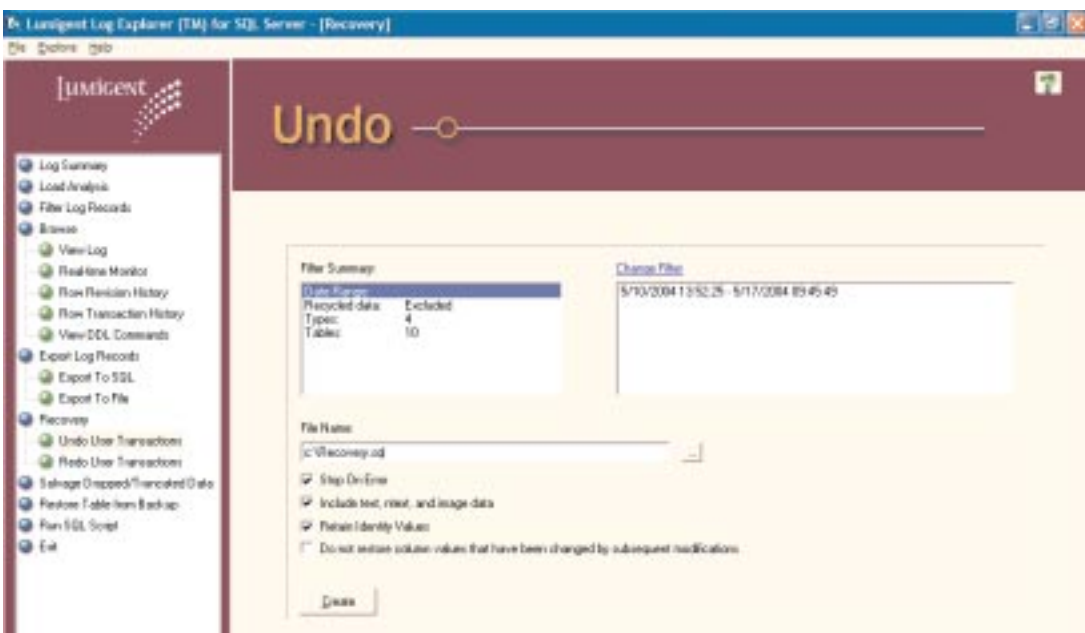
most recent 20 minutes worth of work, you can use Log Explorer to recover the individual transaction. Log Explorer will read the transaction log, find the problem transaction and generate a script to recover the row to its original state.



The first step is to connect to the server and do investigative work to find the transaction. You can do this by specifying the filter by clicking the Change Filter button in the Undo User Transaction screen as shown in Figure 5. Once in the filter, you can filter on a number of items like user name, time, table, or the activity that was performed. Once you have narrowed down

the log, specify where you'd like to output the SQL script to and click Create. This will create an undo SQL script that you can run interactively in Query Analyzer or by clicking Run SQL Script in Log Explorer. My personal preference is to run it in Query Analyzer. That way you can delete any further transactions you'd like to not run.

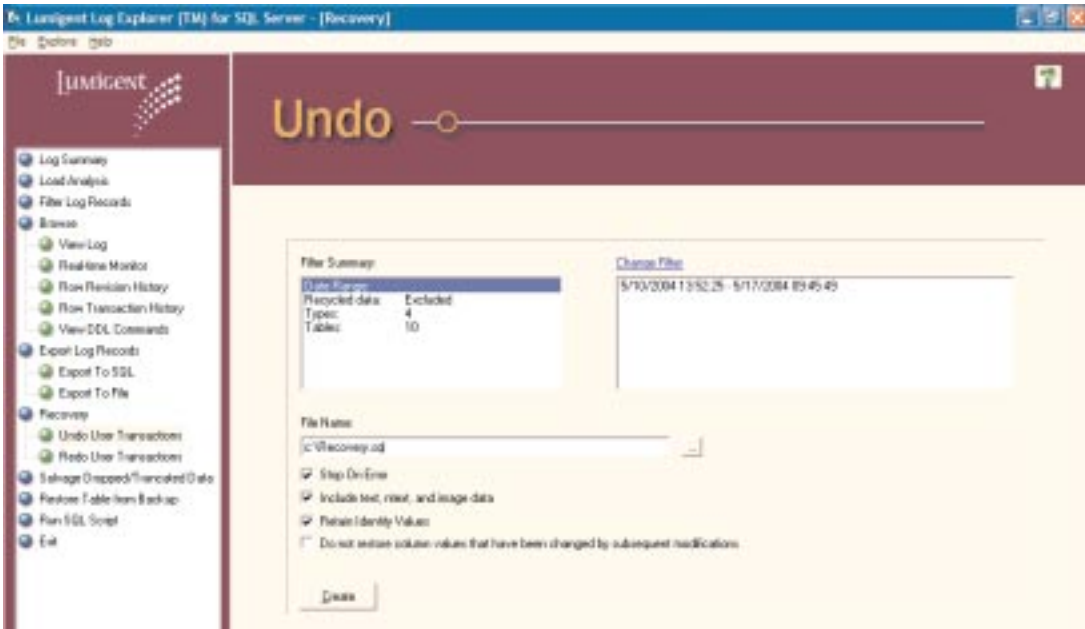
FIGURE 5 UNDOING A USER TRANSACTION IN LUMIGENT LOG EXPLORER



RESTORING DROPPED AND TRUNCATED TABLES

Another scenario you sometimes see is when a developer deletes or truncates a table in development or a DBA accidentally does this in production. To recover this in Log Explorer, simply go to the Salvage Dropped/Truncated Data screen after connecting to the target server and specify what table you'd like to recover. Next, specify the approximate time when the table was dropped or truncated and the script file name. Finally, you'll need to select whether the table was dropped or truncated then click Create.

FIGURE 6 RESTORING A DROPPED TABLE IN LUMIGENT LOG EXPLORER



This will output an easy to read script much like the one below. My table was very simple but it works just as well with a complex table.

```
CREATE TABLE [dbo].[NewTable] ([Column1] INT)
insert into [dbo].[NewTable] ([Column1]) values (1)
insert into [dbo].[NewTable] ([Column1]) values (661)
```

Another important feature in Log Explorer is the table recovery feature. This feature allows you to recover an individual table from a SQL Server backup. This operates much the same as the last example but from a backup instead of the live transaction log.

CONCLUSION

Disaster recovery is one of those necessities that no one thinks about until it's too late.

Most DBAs may not even know their job depends on it until a disaster happens and that sickening feeling hits their stomach. If there's one thing

to take away from this, it's that there's

no such thing as too much planning

and documentation when it comes

to your company's database DR strategy. Good luck

in formulating your strategy and I hope that the scripts

that have been provided aid in your recovery time.



Copyright © 2004 Lumigent Technologies, Inc.

All rights reserved. Lumigent, Log Explorer and the Lumigent Logo are trademarks or registered trademarks of Lumigent Technologies, Inc.