# Technical report:

# Microsoft SQL Server and IBM System Storage N series with SnapDrive

*Integration Guide*

*Document NS3247-0*

September 24, 2007

## Table of contents

## Abstract

*IBM System Storage N series with SnapDrive integrates Windows OS-based applications with IBM N series storage systems. This tight integration makes it quick to implement Microsoft SQL Server solutions on IBM N series storage systems and to easily utilize IBM N series enterprise-class file storage solutions. SnapDrive 2.0 is the second generation of the SnapDrive product with enhanced functionality and a more user-friendly integration utilizing a single storage system's volume for multiple devices. This guide describes how to integrate SnapDrive 2.0.1 with SQL Server and how to utilize IBM N series storage system functionalities such as IBM System Storage N series with Snapshot, IBM System Storage N series with SnapRestore, and IBM System Storage N series with SnapMirror.*

## Introduction

IBM® System Storage™ N series with SnapDrive® has evolved to utilize more I/O protocols and integrates with more IBM N series filer features, but even so, it is extremely easy to integrate Microsoft® SQL Server with IBM N series storage system technology. SnapDrive 2.0.1 use of qtrees has increased the flexibility when multiple devices utilize a single IBM N series storage system volume. Support for asynchronous mirroring has made it much easier to create a robust SQL Server environment for disaster recovery. Support of the new Fibre Channel (FC) I/O protocol has drastically increased the I/O throughput SnapDrive can support and it has made it easier to integrate into existing data-center infrastructures, but it has also made it more complicated to determine which I/O protocol to implement. The different protocols will be discussed when appropriate, but the focus of this document is not the I/O protocols.

SnapDrive manages virtual disks that use different I/O protocols that functionally don't influence SQL Server in any way; therefore, the term "SnapDrive" or "device" will be used whenever the subject matter is not dependent on the I/O protocol. Only if the behavior changes with the protocol will the specific protocol be mentioned.

This technical report will describe all integration aspects influenced by SnapDrive 2.0.1, its functionalities, or supported virtual disk I/O protocols. How to install SnapDrive will not be described in this report (see the section entitled "Migrating databases" in this document, for SnapDrive installation resources) nor will it describe how to install SQL Server, which is described in SQL Server product documentation. This report will focus on integration of SQL Server with the use of virtual disks that are stored on the storage system, IBM System Storage N series with Snapshot™ creation of these virtual disks and restoring these virtual disks from a snapshot, multiple virtual disks utilizing one single volume, and how to utilize IBM System Storage N series with SnapMirror®. Use of virtual disks as Microsoft Cluster Services physical disk resources will also be discussed in this document.

## Supported SQL Server versions

SnapDrive 2.0.1 or newer is application independent and all SQL Server versions are supported. Supported Microsoft Windows platforms are described in the section entitled “Migrating databases” in this document.

## Installing SnapDrive 2.0.1 and SQL Server

Installation of SnapDrive is straightforward and has few special requirements. Detailed descriptions of the installation process of SnapDrive 2.0.1 can be found in the section entitled “Migrating databases” in this report. Installation of SQL Server is described in the SQL Server reference set delivered with the SQL Server CD or see the section entitled “Installing SnapDrive 2.0.1 and SQL Server” in this report.

## Dependencies

There are no dependencies among SQL Server, its installation process, or any SnapDrive components. Therefore, the installation of SQL Server and SnapDrive components can occur independently from each other. If the goal is to install SQL Server’s system database on virtual disk, then it is easiest to install SnapDrive and create the virtual disk before installing SQL Server. It is easiest to specify the storage location during the installation process for all system databases; it is not possible to specify different locations for each system database. Each system database can be moved individually to a new storage location after installation (see the section of this report entitled “Migrating databases” for more details). SnapDrive 2.0.1 is supported under Windows 2000 Server or Advanced Server (while not available at the time of this report writing, SnapDrive 2.0 and later versions also are supported under Windows 2003).

### Starting and stopping SQL Server

The recommendation is to locate all SQL Server system databases on a virtual disk, but special considerations have to be taken when user databases are located both on virtual disks and on other non-IBM N series storage products.

Without access to its master database SQL Server will not be able to start. All global objects are described in the master database, like database names and their paths to the databases’ primary files. Therefore, if the master database is located on an IBM N series virtual disk, but the virtual disk on the storage system is unavailable due to lack of Gigabit or FC network connectivity, then SQL Server will not be able to start. The same is true for tempdb, which is created when SQL Server is started. When Microsoft SQL Server Agent is used, its database msdb has to be accessible or it can’t start up.

## Creating virtual disks using SnapDrive

Prior to the creation of virtual disks using SnapDrive, the storage system environment has to be prepared and SnapDrive has to be installed according to the recommendations described in the section entitled "Migrating databases." The main difference between the supported SnapDrive I/O protocols is the transport layer and the I/O protocol.

Creating virtual disks using SnapDrive requires only a few steps after the system environment has been set up and configured correctly.

### Volume creation and sizing

Creation of storage system volumes is described in the section entitled "Migrating databases"; this document will briefly mention areas related to sizing a volume for SQL Server application or basic performance implications derived from the physical layout. The main issue that has to be decided is the number of disk devices in a volume and the number of data files used by a database table. You must consider the following when sizing a volume:

- The initial database size is the initial starting point for any sizing project.
- The database growth rate is important for sizing the volume(s) correctly or the volume(s) will need to be expanded too soon and too often.
- The change rate of the current database is important for estimating the snapshot space requirements.
- Free space requirements for snapshots are at least equal to the size of the virtual LAN drive (VLD) or logical unit number (LUN) plus changed file blocks.
- The recommendation is to use the default redundant array of inexpensive disks (RAID) group size.
- It is also recommended to operate with extra free volume space—maintaining extra free volume space will decrease a possible volume fragmentation rate and will also prevent sudden "out of space" events; also, when a volume's free space is very limited then I/O performance could suffer.
- Depending on the access pattern, a single disk drive can support about 150 I/O operations per second (IOPS) with current disk technology; therefore, it is important to understand the I/O rate during peak load to estimate the number of disk drives needed to support the I/O load—a RAID group size of eight disk drives (seven data and one parity) can at most support 7 * 150 IOPS = 1,050 (4KB) IOPS.

This is an example of a fictive customer with the following requirements:

- Initial database size is 100GB.
- Database growth rate is estimated to be 10% per month.
- Change rate of the current database is estimated to be 15% per month.
- Snapshot requirement is four snapshots per day with a total of 12 snapshots (three days).
- Default RAID group size is 72GB * 8 devices using RAID 4.
- The administrator wants to expand the volume only every six months.
- The growth and change rates are estimates, so extra volume space has been requested, and the customer realizes that a volume always has to operate with some free space to decrease the possible fragmentation rate, or I/O performance could suffer; therefore, an extra 20% free space per disk drive will be allocated as a free-space buffer.
- The average I/O rate is about 1.5MB per second and the peak rate is about 3MB per second.
    1. The database size after six months will be about 180GB.
    2. About 2GB of the database will change every month after six months, which is equal to 0.15GB per four hours.
    3. The minimum space requirement after six months will be (180GB * 2) + (0.15GB * 12) = 362GB.
    4. A 72GB disk drive has 68GB usable file space, and since 20% is allocated as extra free space, only 55GB is usable per disk drive. Hence, six disk drives are needed for data and one disk drive for parity (RAID4), for a total of seven disk drives. However, it is important for performance reasons to always configure complete RAID groups; therefore, the volume will be created with eight disk drives. If the rate of growth is consistent over time then the volume will have to be expanded with another RAID group after six or seven months.
    5. The estimated peak load was 3MB per second, which is equal to about 770 IOPS.  Since seven data disk drives can support 1,050 IOPS, a RAID group size of eight will be sufficient to support both space requirements and I/O load requirements.

The calculation didn't include the size of system databases. Except for tempdb, all system databases are fairly small. Tempdb is a work area for all SQL Server's temporary storage requirements. The size of tempdb depends exclusively on the operations executed by database applications, number of databases, and number of users, and it can become very large. It is very hard to predict how big tempdb can become, but the I/O load can become very high. Since it is re-created every time SQL Server starts up and is never reused, the tempdb shouldn't be located in a volume where snapshots are created, but on another volume with no snapshots.

## Create a virtual disk

The Installation and Administration Guide for SnapDrive (see the section entitled “Migrating databases” in this report) explains in detail how to create virtual disks. This section will discuss areas that are in some way related to SQL Server functionality.

The main issues are:

- How large should the virtual disk be?
- How much space should be reserved for snapshots? (A detailed description of space reservation is found in the section entitled “Migrating databases”)
- How many devices should be created?
- How much space is needed for the initial database size?
- How much space is needed for future growth?
- Should SQL Server’s system database be located on a storage system device or on local devices?

### Sizing the virtual disks to be created

Storage requirements were calculated in the section of this report that is entitled “Volume creation and sizing”; use this size for sizing the device. One question that has to be answered is how many files to spread the database over and in how many devices. The answer in most cases is one data file and one log file in one device. But in some cases it will make sense to use several data files in one or several devices.

### When to use more than one data file and device

If the database application is an OLTP application then the access pattern is mostly random access and one single data file is appropriate, except if the database is big, in which case the bigger tables ought to be spread over several data files in one SQL Server defined file group (see the section of this report entitled “Database using file groups”). There is no objective answer to when to use one single file in a file group or several files; it mostly depends on I/O rate, sequential access, and access pattern (see the section of this report entitled “Physical layout of databases” for more details). All data files in a SQL Server file group can be located in one single device, but if the application generates a high load of serial read I/O requests then the performance would probably increase if the data files were located in several devices. A data warehouse application has high serial read requirements and is a good candidate for using file groups with several devices.

Another good reason for using more than one data file in several devices is if the database consists of several tables and some of the tables almost never change while other tables change all the time, in which case it make sense to locate tables with a high rate of change in a file(s) located in one device utilizing a volume and the other more static tables in another device utilizing another volume.

Two excellent references for capacity planning and performance tuning are sections of this report entitled “Dependencies” and “Creating virtual disks using SnapDrive.”

### Reserve space for snapshots

If snapshots will be used for backup or other purposes, space has to be allocated for snapshot creation. Free volume space equal to the size of the virtual disk has to be available for SnapDrive to create a snapshot. A detailed description of space reservation is found in the section of this report entitled “Migrating databases.”

It is easy to expand the volume if there isn’t enough free space to allow for snapshot creation as long as hot-spare disks are available on the storage system. Remember to expand the volume in accordance with the RAID group size; otherwise performance could suffer severely.

## Using VLD

VLD is implemented in software; it uses Gigabit Ethernet transport layer and CPU power for all its processing requirements. VLD is an excellent solution when the goal is to implement a stable, reliable environment when high-end I/O performance is not a major requirement. VLD can sustain fairly high I/O throughput and excellent service times, but a FC–connected LUN is able to deliver higher performance.

## Using fiber channel-connected LUNs

LUN uses FC infrastructure and does not use the CPU power as much as VLD. In system environments where I/O performance is a major requirement, LUN is an excellent choice; it delivers high-end I/O throughput and servicing times. LUN is as stable and reliable as VLD and will deliver greater maximum I/O throughput and shorter I/O latencies.

## Converting VLD to LUN

It is straightforward to convert a VLD device to LUN, but it is a one-way conversion and can’t be reversed. The conversion consists of three steps:

1. Disconnect the VLD device. In SnapDrive Manager, highlight the VLD to convert and right click. Select actions disconnect.
2. Convert the VLD to a LUN. From the right panel select and highlight SnapDrive. Select action “Convert VLD to LUN…” and follow the instructions.
3. Connect to the LUN. Highlight SnapDrive and select action “Connect disk…” and follow the instructions.

## VLD or LUN

LUN is for system environments with high-end performance requirements, but the decision is often based on which transport layer is already in place at the customer site. Most customer sites already have some Gigabit Ethernet infrastructure configured and available for the new SQL Server environment. Even when a new Gigabit Ethernet infrastructure has to be created, it is very cost efficient and reliable, and it will in most cases satisfy system requirements. If a FC infrastructure is already in place and it can be used with IBM N series SAN technology then it makes sense to use the LUN technology. In cases where the performance requirements are not well understood, start by testing with VLD and if that environment doesn’t fulfill expectations then convert to LUN.

## Database creation

Device size(s) and therefore the size(s) of the NT file system(s) (NTFS) is already known from the material presented in the section of this report entitled “Create a virtual disk.” In this section we will discuss how many log and data files to use for the database and how to create a database. Databases can be created from either Query Analyzer (using T-SQL) or from Enterprise Manager (a SQL graphical interface tool); both possibilities will be illustrated.

SQL Server views all log files belonging to a database as one single logical log file. This doesn’t change with the number of physical files the log consists of. The only reason to have more than one log file is if a single device is not big enough to contain the complete log file; therefore, smaller files on different devices have to be used. SQL Server will use only one physical file at a time; when all space in one file is used then SQL Server will move on to the next file. Data files are different: when the SQL Server database engine detects that a table or index is located on several files in a file group it will use that knowledge to concurrently read from several files at the same time, which is the case when a table or index has to be scanned.

Section “Create a Virtual Disk” discussed in general terms sizing requirements for a database but it did not discuss how many data files to use and how many devices. It is very hard to give concrete guidelines; most known customer cases use just one single data file for a database. Using more data files and storage system groups makes the database layout more flexible, and it will make it easier to move part of a database to another device for performance reasons. This will especially be true in the future, when the database and the performance requirements will, most likely, be very different compared to today. As the database size increases the I/O characteristic changes and so does the transaction rate. Therefore, flexibility and a flexible physical layout are important.

Each database table object or index object can be located in its own file group. A file group is a logical grouping of physical files. A file in a file group can’t be moved to another file group, but files can be added to a file group. A table, or index, that is created in a file group with more than one data file will be allocated in a round-robin fashion across all files in that file group relative to each file’s size. Hence, it is important with an IBM N series storage system to create all files with equal sizes. When a database’s physical layout utilizes several file groups and if in the future the storage is overutilized resulting in poor performance, then it will be fairly easy to move a heavily loaded file group or to move one of its files to another volume.

## How much space is needed for the database files?

The initial space and future growth requirements have to be known at the time of volume(s) creation; of more concern at this point is the size of the database files. It is necessary to first clarify the difference between extending a database table space and growing a database file. A table is always extended in fixed allocation sizes (eight database pages) and can’t be changed. When a table has to be extended and there is not enough free file space then the file will grow if the “automatically grow file” property is configured. In other words, tables will extend on already allocated and initialized database files whereas files will automatically physically grow and initialize database files when there is no more free space to extend tables. File growth can be set to fixed allocation sizes or a percentage relative to the current size. When growth size is set to a percentage the actual growth size will increase over time as the file size increases, which can become a problem with very large files. A fixed growth size is easier to control and can always be changed through Enterprise Manager.

Data files contain table data and/or table indexes, whereas log files contain transaction log records. The size of a log file depends mostly on transaction arrival rate and the database’s recovery mode. A SQL Server database can be in one of three recovery model types:

- The full recovery model provides the least risk of losing transactions in the case that the backup has to be used for recovering a database because of lost media.
- Bulk logged doesn’t fully log certain bulk operations (like BULK INSERT, BCP, and CREATE INDEX); only minimal information is logged in the log file. The minimal logging can still fully recover the database. The tradeoff is that bulk operations execute faster compared with the full recovery model, but it will take longer to back up the log file because all database pages modified by a bulk operation have to be backed up together with the log file. Hence, the transaction log is smaller but the backed up transaction log is much bigger.
- Simple means that logging of transactions are disabled and only full or differentiable backups can be used (no backup of the transaction log).

When full or bulk logged is used then the transaction log file will grow until either there is no more free space on the log device or the log file has reached its maximum size as defined during database creation, in which case transaction processing will stop until the log file has been backed up or more space has been allocated to the device. After the transaction log has been backed up then it is logically truncated and the space can be reused for new transaction records.

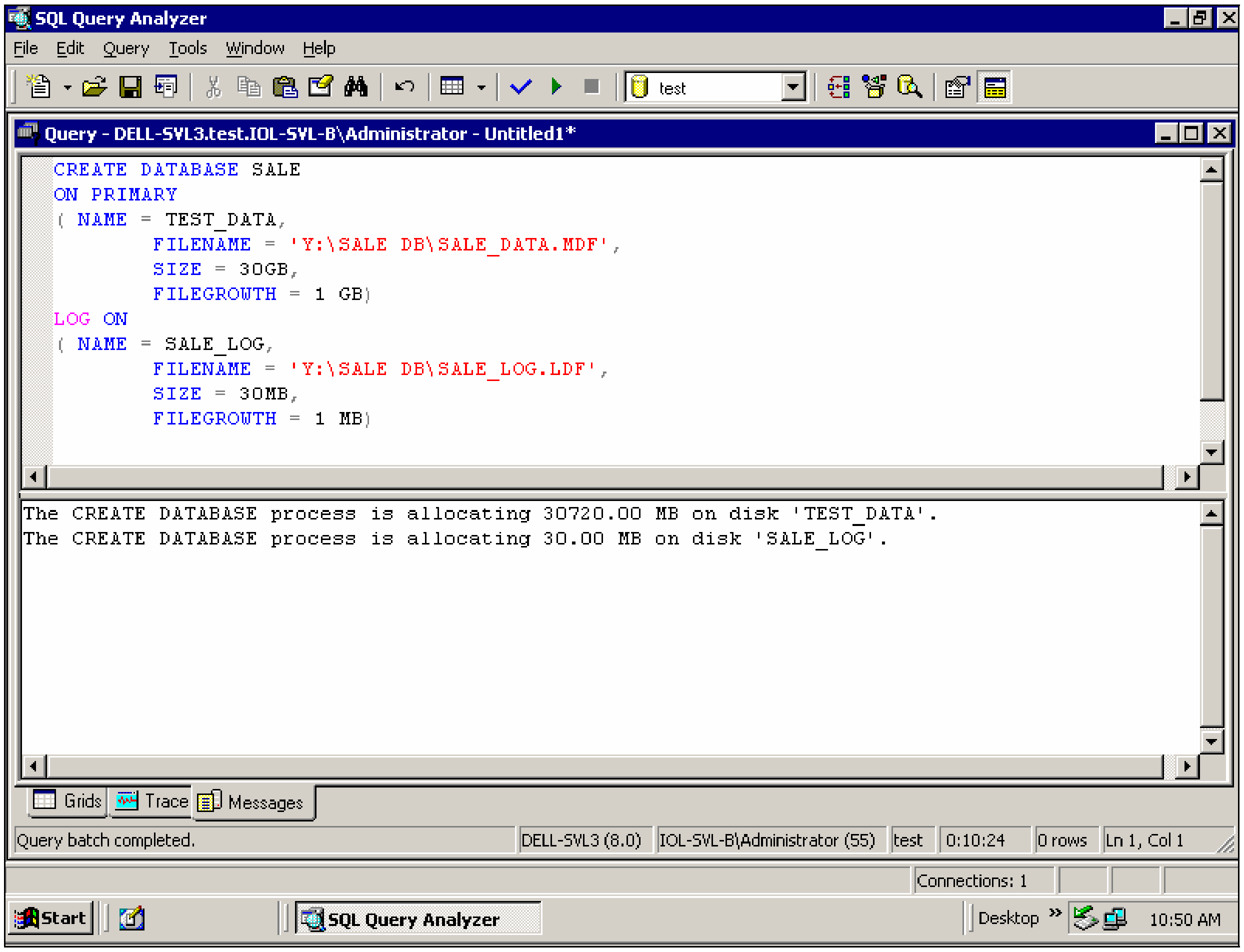
When space is estimated for a database’s transaction log, it has to be based on the recovery mode, transaction arrival rate, and how often the log is backed up if the recovery mode is either full or bulk logged.

## Creating a database using T-SQL and Query Analyzer

Two different database creation examples will be given. The first example will create a basic database with one data file and one log file. The second example will create a more complex physical layout of a database with several file groups and data files. The storage protocol can be VLD or FC; this does not affect the creation of a database.

## Create database with one file group

Figure 1 shows Query Analyzer's execution and result screens. The CREATE DATABASE statement creates a SALE database consisting of one data file in the PRIMARY file group. Both the data and log file can grow unlimited until the device is full.  Both the data file and the log file are located on the "Y" device.


SQL Query Analyzer
File Edit Query Tools Window Help
test
Query - DELL-SVL3.test.IOL-SVL-B\Administrator - Untitled1*
Grids
Trace
Messages
Start
SQL Query Analyzer
Desktop
10:50 AM

```
CREATE DATABASE SALE
ON PRIMARY
( NAME = TEST_DATA,
        FILENAME = 'Y:\SALE DB\SALE_DATA.MDF',
        SIZE = 30GB,
        FILEGROWTH = 1 GB)
LOG ON
( NAME = SALE_LOG,
        FILENAME = 'Y:\SALE DB\SALE_LOG.LDF',
        SIZE = 30MB,
        FILEGROWTH = 1 MB)
```

```
The CREATE DATABASE process is allocating 30720.00 MB on disk 'TEST_DATA'.
The CREATE DATABASE process is allocating 30.00 MB on disk 'SALE_LOG'.
```

```
Query batch completed.    DELL-SVL3 (8.0)    IOL-SVL-B\Administrator (55)    test    0:10:24    0 rows    Ln 1, Col 1
Connections: 1
```

*Figure 1. Create Database with Query Analyzer*

## Database using file groups

This example creates a database named SALES with three file groups and one log file:

- The primary file group with the files named Spri1_dat and Spri2_dat; both files' initial file sizes are 100MB with a growth size of 10MB; both files are located on the same device.
- A file group named SALESGROUP1 with the files SGrp1Fi1 and SGrp1Fi2; the file's initial size is 150GB with a growth size of 5GB; the two files are located on two different devices, "Y" and "Q."
- A file group named SALESGROUP2 with the file named SGrp2Fi1, which is located on its own "X" device.
- The log file is located together with the primary file group on device "Z"; the maximum size of the log file is 150MB with an initial size of 50MB and a growth size of 5 MB.

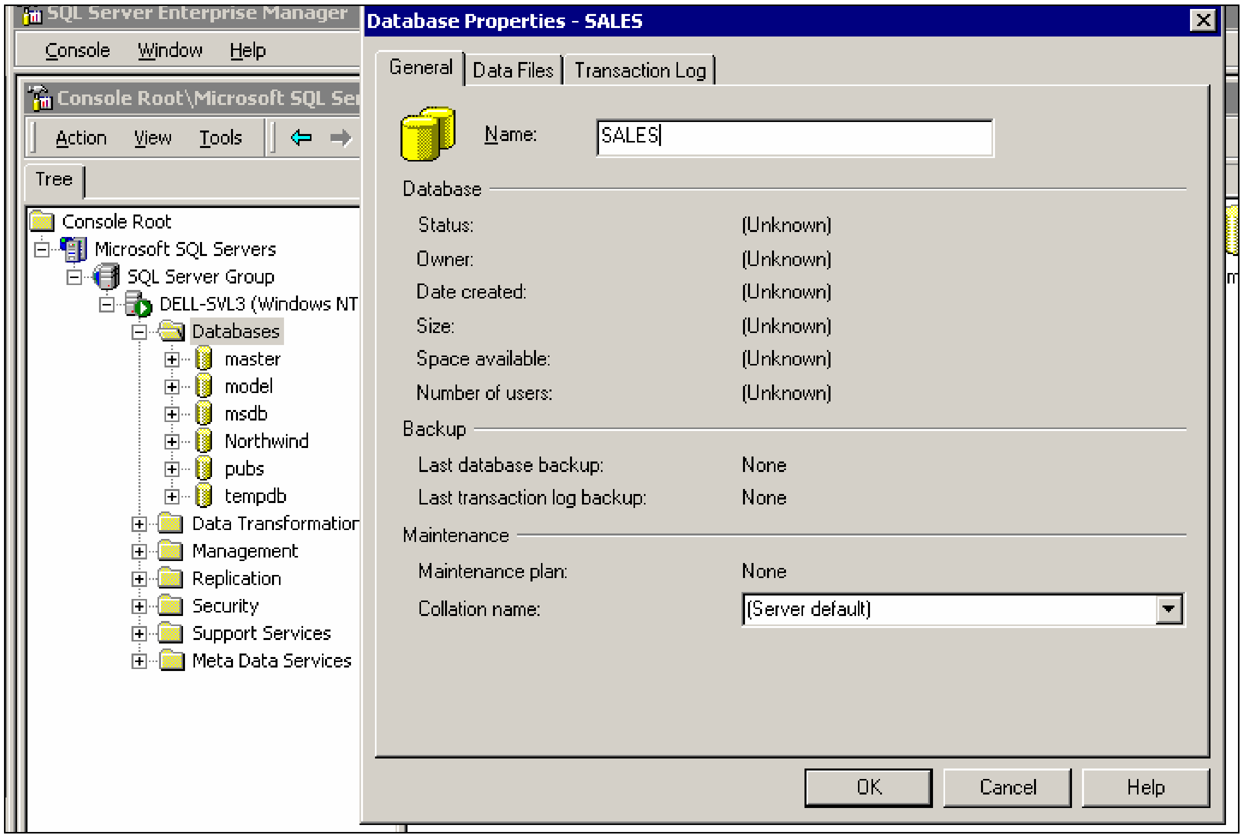
The physical layout of a database and the reasons to use several devices and files were discussed in sections “Volume Creation and Sizing” and “Create a virtual disk.”

```
USE MASTER
       GO
CREATE DATABASE SALES
ON PRIMARY
(NAME = SPRI1_DAT,
                   FILENAME = 'Z:\\MSSQL\SPRI1DAT.MDF',
                   SIZE = 100MB,
                   FILEGROWTH = 10MB),
(NAME = SPRI2_DAT,
                  FILENAME = 'Z:\ MSSQL\SPRI2DT.NDF',
                   SIZE = 100MB,
                   FILEGROWTH = 10MB),
FILEGROUP SALESGROUP1
(NAME = SGRP1FI1_DAT,
                   FILENAME = 'Y:\MSSQL\SG1FI1DT.NDF',
                    SIZE = 150GB,
                    FILEGROWTH = 5GB),
(NAME = SGRP1FI2_DAT,
                    FILENAME = 'Q:\MSSQL\SG1FI2DT.NDF',
                    SIZE = 150GB,
                   FILEGROWTH = 5GB),
FILEGROUP SALESGROUP2
(NAME = SGRP2FI1_DAT,
                  FILENAME = 'X:\MSSQL\SG2FI1DT.NDF',
                   SIZE = 1 GB,
                    FILEGROWTH = 50MB)
LOG ON
( NAME = 'SALES_LOG',
                   FILENAME = 'Z:\MSSQL\SALELOG.LDF',
                   SIZE = 50MB,
                   MAXSIZE = 150MB,
                   FILEGROWTH = 5MB )
`      GO
```

## Creating a database using Enterprise Manager

An alternative to using a T-SQL statement for creating a database is Enterprise Manager. It is fairly easy to use, and it is not necessary to know either Query Analyzer or T-SQL.

Start Enterprise Manager and browse to Databases in the right panel. Right-click Databases, select New Databases, and name the database to be created in the General tab (Figure 2).



Name: SALES

Database

| | |
|---|---|
| Status: | (Unknown) |
| Owner: | (Unknown) |
| Date created: | (Unknown) |
| Size: | (Unknown) |
| Space available: | (Unknown) |
| Number of users: | (Unknown) |

Backup

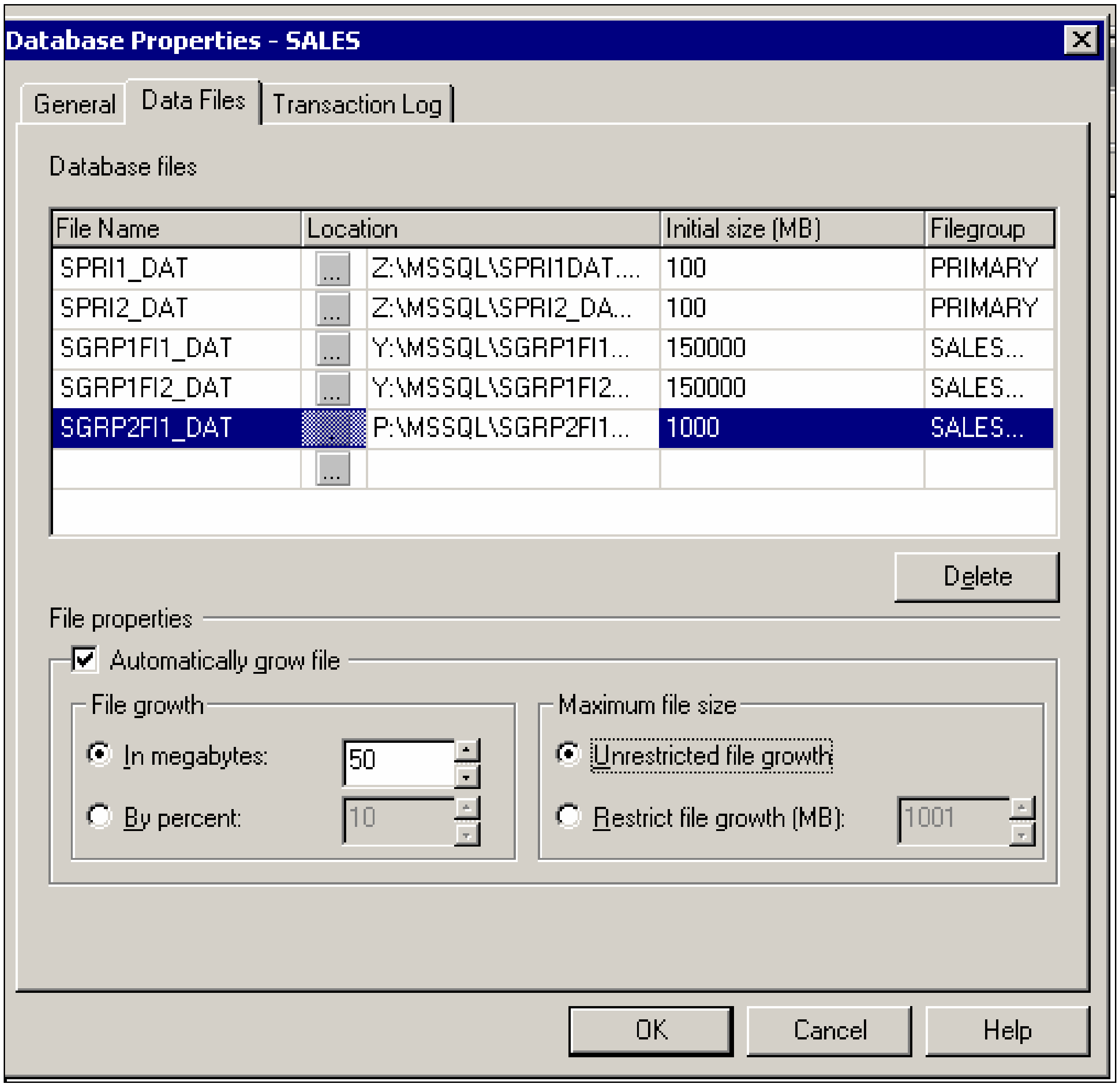
| | |
|---|---|
| Last database backup: | None |
| Last transaction log backup: | None |

Maintenance

| | |
|---|---|
| Maintenance plan: | None |
| Collation name: | (Server default) |

*Figure 2. Create Database General Tab*

Go to the Data File tab and fill in all information related to all data files. File growth is by default set to 10% auto increase; so remember to change it if a fixed growth size will be used (Figure 3).



Database Properties - SALES

General | Data Files | Transaction Log

Database files

| File Name | Location | Initial size (MB) | Filegroup |
|---|---|---|---|
| SPRI1_DAT | Z:\MSSQL\SPRI1DAT.... | 100 | PRIMARY |
| SPRI2_DAT | Z:\MSSQL\SPRI2_DA... | 100 | PRIMARY |
| SGRP1FI1_DAT | Y:\MSSQL\SGRP1FI1... | 150000 | SALES... |
| SGRP1FI2_DAT | Y:\MSSQL\SGRP1FI2... | 150000 | SALES... |
| SGRP2FI1_DAT | P:\MSSQL\SGRP2FI1... | 1000 | SALES... |

File properties

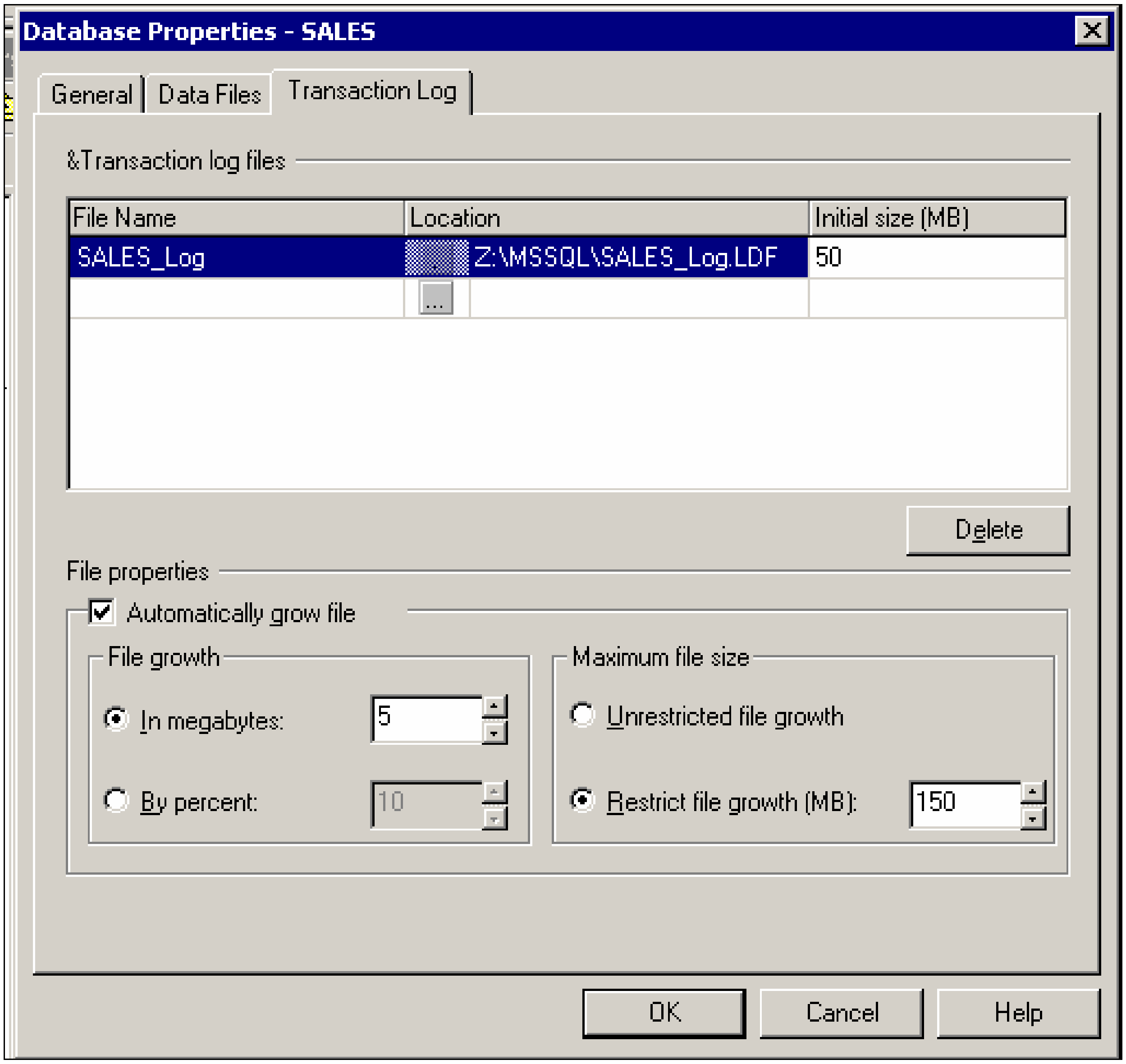
[✓] Automatically grow file

File growth
- (●) In megabytes: 50
- ( ) By percent: 10

Maximum file size
- (●) Unrestricted file growth
- ( ) Restrict file growth (MB): 1001

*Figure 3. Create Data File Tab*

Figure 4 shows definition of the transaction log file. (The log file doesn’t belong to any file group.)



Database Properties - SALES

General | Data Files | Transaction Log

&Transaction log files

| File Name | Location | Initial size (MB) |
|---|---|---|
| SALES_Log | Z:\MSSQL\SALES_Log.LDF | 50 |

File properties

[✓] Automatically grow file

File growth

[● In megabytes:] 5

By percent: 10

Maximum file size

Unrestricted file growth

[● Restrict file growth (MB):] 150

*Figure 4. Create Transaction Log Tab*

## Physical layout of databases

Most SQL Server environments will perform to satisfaction when all database files are located in one single volume. However, some environments that require high performance levels need dedicated attention to the physical layout of the database files. The physical layout is influenced by the database type and the I/O load generated by the user of the application. This paper distinguishes between applications generating mostly random database access (OLTP, online transaction processing) and applications that generate mostly sequential database access (data warehouse). The database creation examples in 7.1.2 and 7.2 show a physical layout over three devices and the possibility for tables and/or indexes to be allocated over two files in different file groups.

### Physical layout of OLTP databases

OLTP types of applications usually execute workloads with mostly random access patterns to the database files and read mostly one database page at a time. Most OLTP applications also execute some very limited sequential access. A high level of sequential reads usually means that an index is missing or created incorrectly or the application is not a “clean” OLTP application.

The main performance goal of an OLTP database is to limit disk access latency by physically distributing the database over as many disk drives as possible and not to have any I/O hot spots.

It is easiest to locate all files belonging to a database in one single volume (see the use of snapshots in the section of this report entitled “Using snapshots with SQL Server 2000”). If initial testing of a database shows performance problems then begin by moving the transaction log file to another volume. Additionally, if tempdb is located in the same volume as the user database, then move tempdb to another volume. If that doesn’t help then locate the data files in different volumes (see 6.1 and 6.2), which is especially important if a database table is very big.

### Physical layout of data warehouse databases

Data warehouse applications usually generate a database I/O load that consists mainly of sequential I/O read access. Distributing a database table over several files in file groups will enhance the possible maximum read throughput. How much the throughput will increase depends on the application’s access pattern, but allocating a table over several files will in general improve the performance.

## Migrating databases

When an IBM N series storage system is introduced into an existing SQL Server environment then databases have to be migrated from the current storage location to the new storage system storage. It is not complicated to migrate system databases to the new storage, but different methodologies have to be used for each system database. Tempdb is critical to the performance of all SQL Server's databases, and it is recommended to move this system database to a storage system volume. The migration described in this section is concerned only with migrating from a third-party disk subsystem or from a universal naming convention (UNC) path.

### System databases

SQL Server has five system databases:

- The master database is the main database of all SQL Server system databases; many activities modify the content of the master database, for example, information about user-created databases such as their location, creation, alteration, and removal.
- The model database is used as a template when user databases are created; the system administrator can change the properties of the model database to adapt it to the specific needs of the local database environment.
- The tempdb is used for temporary storage and is re-created every time SQL Server is started.
- Information needed by the SQL Server Agent is stored in msdb, e.g., alerts and jobs.
- The distribution database is used for replication and will be visible only if replication is active.

#### Moving System Databases

It is fairly easy to move system databases, but different methodology is needed for each system database.

##### Master

- There may be reasons to move the master database, but one has to be careful not to corrupt the entire SQL Server environment; moving the master database involves two steps: assigning the new location within the SQL Server configuration and moving the associated files.
- Run DBCC CHECKDB prior to migrating the database.
- From Enterprise Manager, right click the SQL Server and select Properties.
- Click the Startup Parameters button at the bottom of the General properties page.
- There are usually three parameters, two parameters referencing the database files and one parameter referencing the message log location; highlight each of the existing parameters, remove them, and add the new paths.
- Stop SQL Server.
- Move or copy the files to new location.
- Restart SQL Server.

## Tempdb

All temporary information is stored in tempdb, which is re-created each time SQL Server is started.  How much tempdb is used is application dependent, but in general SQL Server generally heavily uses tempdb.

Since tempdb is re-created every time SQL Server is restarted, there is no reason to create snapshots of it; therefore, locate it in a VLD that will not be taken snapshots of.

The database command “alter database” is used for moving the tempdb (see SQL Server Online Help for a detailed description of the “alter database” statement):

- **Move data file.** Alter database tempdb modify file (name=’tempdev’, filename=’newpath\newfilename’)
- **Move log file.** Alter database tempdb modify file (name=’templog’, filename=’newpath\newfilename’)
- **Stop and restart SQL Server.**

## Msdb

The msdb is used by SQL Server Agent to manage alerts, jobs, and scheduling. Follow these steps to easily move msdb:

- Stop SQL Server Agent
- From Query Analyzer detach the database
- USE master
- GO
- Sp_detach_db ‘msdb’
- GO
- Move the data and log files to the new location
- USE master
- Sp_attach_db ‘msb’ (see 7.2)
- GO
- Start SQL Server Agent

### Model

In order to move the model database to a new location, SQL Server has to be started with Trace flag 3608, which prevents recovery of any database except master. Follow these steps:

- From Enterprise Manager, right-click the SQL Server and select Properties
- Click the Startup Parameters button
- Add the startup parameter –T3608
- Stop and restart SQL Server
- Detach the database from Query Analyzer (see 7.2)
- Move the data and log files to the new location
- Reattach the model database (see 7.2)
- Remove the trace flag from the startup parameters
- Stop and restart SQL Server

## User databases

SQL Server functionality makes it easy to move databases from an UNC path to a storage system or third party directly attached disk to IBM N series virtual disk. Detach the database(s) from the undesired location, copy the data files to the new location, and reattach the databases files to the new location. The steps can be implemented from the Query Analyzer as follows:

1. Detach current database files:
```
sp_detach '[database name]'
GO
```
2. Copy files:
```
copy *.MDF   E:    /*  copy data files */
copy *.LDF    E:   /* copy log files */
```
3. Attach new database files:
```
USE master
GO
sp_attach_db '',
    'E:\[folder]\[name].MDF',
    'E:\[folder]\[name].LDF',
GO
```

## Using snapshots with SQL Server 2000

Snapshots have to be created and restored through the SnapDrive Microsoft Management Console (MMC) snap-in or command line interface (SDCLI.exe). This section will describe how to create snapshots of SQL Server databases. The methodology used when creating snapshots depends on several factors: number of databases, database stored on one or several devices over how many volumes, or whether the database engine can be shut down for a short duration. Snapshot creation depends on SnapDrive functionality, which has to be well understood.

A snapshot can only be used as a point-in-time backup and recovery.

### Creating snapshots with SnapDrive 2.0.1

Creating snapshots depends on SnapDrive functionality. SnapDrive manages snapshot creation of virtual disks. The snapshot creation takes place on the storage system. Snapshots are created per volume, but SnapDrive prepares individual virtual disks for snapshot creation. SnapDrive will not recognize virtual disk snapshots that are not created by SnapDrive. In addition, snapshot creation of virtual disks must be initiated by a SnapDrive instance that is running on a Windows host that owns the virtual disks.

SnapDrive creates snapshots of a volume and not a device, whereas SnapDrive restores a VLD or LUN device and not the volume. It is essential to understand the effect this has when a database utilizes several devices in one single volume. When a snapshot is created of a device in a volume, all other devices will be snapshot at the same time. This is not a big issue but it is important to understand that all devices used by a single database have to be restored at the same time; if one device is not restored the database will probably not be able to go online because of inconsistencies.

### SQL Server mode and database modes

As long as only one volume is utilized by a database, it doesn't matter which mode SQL Server is executing in or if the database is online or offline; snapshots can just be created and SQL Server will crash recover after devices (VLD or LUN) have been restored using IBM System Storage N series with SnapRestore®. When the database utilizes more than one volume, then special care has to be taken. Before planning how to create snapshots one has to understand the different SQL Server execution modes. It is preferable to stop SQL Server or to take the database(s) offline while creating snapshots. If the database(s) utilizes more than one device in more than one volume, special care has to be taken:

- The normal execution mode for SQL Server is active, multiuser mode, and databases are online; new transactions can arrive and be committed at any time.
- When SQL Server is stopped, all databases are checkpointed and all users are disconnected.
- When a database is in single-user mode, then only one single user can access the database; that user can still execute new transactions.
- A database that is taken offline is checkpointed in the process, and the database can't be accessed while it is offline.
- While a database is online, users can access it and execute transactions.

The different SQL Server execution modes are necessarily to understand what discussing SQL Server's crash recovery in the next section.

## SQL Server crash recovery

When SQL Server stops, normally a checkpoint is executed for all databases. A checkpoint is implicitly executed when a database is taken offline. A checkpoint synchronizes a database's data files with the in-memory images, and a checkpoint marker is written to the log file and data files. If SQL Server suddenly crashes, the data files are not synchronized with the memory image, and there are both committed and uncompleted transactions in the log file. The uncommitted transactions must be rolled back and committed transactions must be rolled forward, which is done during SQL Server's crash recovery.

SQL Server's crash recovery processing logically executes the following steps:

- Find the last checkpoint in the database's log and data files.
- Roll back all open transactions that have not been committed.
- Roll forward all committed transactions.
- Write a synchronization marker to all files.

The first point is central to understanding how to create snapshots while a database is online and utilizes more than one volume. Hence, if a checkpoint is executed while a set of snapshots is being created, SQL Server will not be able to crash recover since the marker for the last checkpoint in the log file is newer than the marker in the data file(s). Therefore, it is crucial to control how often a checkpoint is created, which can be done with the recovery interval database settings that are set from Enterprise Manager.

## Snapshots of databases in a single volume

As long as a database utilizes only one volume, then it doesn't really matter which mode SQL Server is in or if the database is online or offline, one can just create a snapshot. After a SnapRestore SQL Server will crash recover the database.

## Snapshot of databases using virtual disks in several volumes

If the customer environment allows for taking the database offline or stopping SQL Server while snapshots are created, that is the preferred methodology. But this is often not possible and snapshots have to be created while the database is online. In this case, great care must be taken to prevent the database engine from running any checkpoints while snapshots are created. The database administrator (DBA) has to be certain how to control the frequency of running checkpoints and how to create the snapshot of the VLD containing the log file last. It is fairly easy to implement this functionality in a batch program. Execution of the batch program can automatically be executed from SQL Server Agent.

SQL Server's recovery interval together with the arrival rate of new transactions (transactions in the log file) determines how often checkpoints occur. Set SQL Server's recovery interval long enough to guarantee that no checkpoint will be executed while the snapshot is being created. One of SQL Server's performance counters is a good tool to monitor how often a checkpoint is executed. From the Performance Monitor add counter "SQLServer:Buffer Manager; checkpoint pages/sec."

The recovery interval can be set from Enterprise Manager: open the console root and highlight and right-click on the Server instance. Click **Database Settings** and adjust the recovery interval. The batch job outlined in the section of this report entitled "CRE_SNAPSHOT.BAT" can easily be expanded to set a very long recovery interval in the beginning of the batch job and reset it after all checkpoints and snapshots have been created.

## Snapshot batch program of online databases

The following batch job begins with executing a checkpoint per database (checkpoints are database dependent), renaming current snapshots, and creating the new snapshot. The isql statement in the batch job executes an isql input file.

For the script to work correctly, the system path has to include the path to SnapDrive Manager's command line interface, default `<driver>:\Program Files\`SnapDrive.

## CRE_SNAPSHOT.BAT

It is easiest to control and to understand if one batch job is created per database.

When several devices utilize one volume then only one single snapshot should be created per volume (see the section of this report entitled "Using snapshots with SQL Server 2000").

```
isql -Usa -P -S<SQL Server> -i cr_checkpoint.sql
SDCLI snap delete –d <drive letter 1> -s <oldest_data_1>
SDCLI snap rename -d <mount point 1> -o <oldest-1_data_1> –n <oldest_data_1  >
SDCLI snap rename –d <mount point 1> -o <oldest-2_data_1> –n <oldest-1_data_1 >
SDCLI snap rename –d <mount point 1> -o <oldest-3_data_1> -n <oldest-2_data_1>
```

// Continue all current snapshots on all mount points

```
SDCLI snap delete –d <mount point 2> -s <oldest_data_2 >
SDCLI snap rename –d <mount point 2> -o <oldest-1_data_2> -n <oldest_data_2>
SDCLI snap rename –d <mount point 2> -o <oldest-2_data_2> -n <oldest-1_data_2>
SDCLI snap rename –d <mount point 2> -o <oldest-3_data_2> -n <oldest-2_data_2>

SDCLI snap delete –d <mount point 3> -s <oldest_log>
SDCLI snap rename –d <mount point 3> -o <oldest-1_log> -n <oldest_log>
SDCLI snap rename –d <mount point 3> -o <oldest-2_log> -n <oldest-1_log>
SDCLI snap rename –d <mount point 3> -o <oldest-3_log> -n <oldest-2_log>
```

// Continue creating new snapshots of all data files and then

```
SDCLI snap create -s <newest_data1> -D <mount point 1>
SDCLI snap create –s <newest_data2> -D <mount point 2>
SDCLI snap create –s <newest_data2> -D <mount point 3>
```

## CR_CHECKPOINT.SQL

\\ Checkpoint all databases that utilize files in any volume utilized by a used device.

```
cre_snapshot.bat

Checkpoint <database>
Go
```

## Configure SQL Server Agent

SQL Server Agent has a scheduler that can execute the batch job in 7.7. The scheduler is flexible and can execute the job as frequently as needed, every day or on certain days. Remember, when SQL Server is stopped, the SQL Server Agent is also stopped; therefore, be certain to also start SQL Server Agent after stopping SQL Server.  From Enterprise Manager:

- Open the Console Root
- Open Microsoft SQL Server
- Open SQL Server Group
- Open the server
- Open Management
- Highlight Jobs and go to New Job
- Update the four tabs: General, Steps, Schedule, and Notification.

## Restoring databases on virtual disks

Before a database can be restored using SnapRestore it has to be taken offline or SQL Server has to be stopped. SnapRestore has to be executed from the SnapDrive MMC snap-in or SDCLI. After a restored database has been brought online or the SQL Server engine has to be started, the database engine will replay all transactions in the log since the last checkpoint. SnapDrive restores a virtual disk from a specified snapshot copy. The virtual disk is the minimum unit of restoration.  All devices utilized by the database to be restored have to be restored before the database is taken online. If one of the devices utilized by the database is not restored then the database will probably be prevented from coming online.

### Attach SQL Server to database in snapshot

SQL Server can utilize databases in snapshots for reporting or for recovering from soft database errors. Before SQL Server can access the database, the devices in the snapshots have to be connected and SQL Server has to attach the database’s files:

- Use SnapDrive MMC snap-in or SDCLI to connect to the device(s) in the snapshot(s).
- Use Query Analyzer (see 8.2) or Enterprise Manager to attach to a database in the mounted snapshot(s); when the database is mounted in read-only mode SQL Server will not be able to replay transactions committed since the last checkpoint, and not all completed transactions after the last checkpoint will be rolled back.

A snapshot of a VLD can be mounted in read-only or read-and-write mode, whereas a snapshot of a LUN can be mounted only in read-and-write mode.

### Connecting virtual disks in snapshot using read and write access

Snapshots can be mounted in writable mode. Snapshots mounted in writable mode are excellent for testing or other short-term use. However, it should never be used as the base for a new database image. All data written to a virtual disk in a snapshot that is connected in read/write mode will be lost as soon as this virtual disk is disconnected. Avoid creating snapshots of a virtual disk that is on a storage system volume where a virtual disk in a snapshot is connected in read/write mode. It is easy to run out of free space if snapshots are created of writable snapshots.

## Disaster recovery and high availability

SnapDrive 2.0.1 actively supports asynchronous SnapMirror and local synchronous mirroring. SnapDrive is integrated with SnapMirror functionality, whereas local synchronous mirroring is storage system-level mirroring transparent to SnapDrive.

## Asynchronous mirroring for disaster recovery

Asynchronous SnapMirror is an excellent solution for disaster recovery of SQL Server databases. The recommendation is to use rolling snapshots (see the section of this report entitled “Migrating databases”) for disaster recovery. Each time a snapshot has been created SnapDrive initiates the replication of the source volume to the target mirrored volume if SnapMirror is turned on. Snapshots in a mirror can be used in read-only mode as explained in section of this report entitled “Restoring databases on virtual disks.” (Setup and configuration of SnapMirror is explained in the section of this report entitled “Migrating databases.”)

## Local synchronous mirroring for high availability

Local synchronous mirror (LSM) is a single storage system-level mirroring and is transparent for SnapDrive and SQL Server. LSM creates a volume consisting of two duplexes where each duplex is an exact copy of the other at any point in time. LSM is one component of a high-availability system environment, which makes it very unlikely that a volume will fail because of multiple disk failures in a duplex RAID group. (For more information about LSM see the section of this report entitled “Using snapshots with SQL Server 2000.”)

## Cluster failover

Storage system cluster failover (CFO) together with LSM is an important component of high availability. CFO safeguards against single storage system head failure and LSM safeguards against double disk failure in a single RAID group. Both CFO and LSM are transparent for both SQL Server and SnapDrive. (For more CFO information see the sections of this report entitled “Supported SQL Server versions” and “Using snapshots with SQL Server 2000.”)

## SnapDrive and MSCS configurations

SnapDrive supports MSCS configurations. The simplest SQL Server configuration with MSCS is an active/passive configuration, which means one active SQL Server instance executing on one of the cluster nodes. If the cluster is configured with two active SQL Server instances executing on each cluster node then this setup is called an active/active configuration. It is also possible to configure several SQL Server instances to execute on each node, which is still an active/active configuration. Possible configurations are summarized below:

- One single SQL Server instance is executing on one cluster node. The other cluster node is only used for failover if the active node fails. This is an active/passive configuration.
- Two SQL Server instances with one SQL Server instance executing on each node. Each node is the failover node for the other node. This configuration is an active/active configuration.
- Multiple SQL Server instances executing on each node is an active/active configuration.

Each SQL Server instance utilizing MSCS has to be located in its own cluster group with all its resources.

The quickest MSCS setup with SQL Server is to follow the basic instructions in the section of this report entitled “Migrating databases” before installing SQL Server. After installing MSCS and testing that both nodes function correctly, install SQL Server:

1. Create a new MSCS-aware device and locate this device in a new cluster group (a SnapDrive option during device creation).
2. Install SQL Server and point to the new device in the new cluster group.
3. If the installation is an active/active configuration then repeat Steps 1 and 2 from the other MSCS node.
4. If several SQL Server instances will be executing on each cluster node then repeat Steps 1 and 2 for each SQL Server instance.

## Trademarks and special notices