SYBASE SQL Server™ Troubleshooting Guide

SYBASE SQL Server Releases 4.2–10.0.2

Document ID: 39998-01-1000-04

Last Revised: September 15, 1995

Author: Sybase Technical Support Publications

Document ID: 39998-01-1000

Document Orders

Sybase Trademarks

Restricted Rights

Table of Contents

List of Tables

Preface

The *SYBASE SQL Server Troubleshooting Guide* describes error conditions and troubleshooting procedures for problems that SYBASE® users may encounter when using SYBASE SQL Server™. The problems addressed here are those which the Sybase Technical Support staff hear about most often. The guide's purpose is:

- To provide enough information about certain error conditions so that you can resolve problems without help from Technical Support
- To provide lists of information that you can gather before calling Technical Support, which will help resolve your problem quickly
- To provide you with a greater understanding of Sybase products

Audience

This guide is intended for the following:

- SYBASE System and Database Administrators
- Sybase Technical Support contacts
- Developers of applications using SYBASE software

This guide assumes that you are thoroughly familiar with the Sybase products. If you are unfamiliar with any of the procedures described in this guide, call Sybase Technical Support for assistance.

What This Guide Contains

This guide contains these chapters:

- Chapter 1, "SQL Server Memory Issues," explains how SQL Server manages memory and how you can configure your SQL Server to use memory most efficiently.
- Chapter 2, "Transaction Log Management," includes information about transaction logs and procedures for managing them.
- Chapter 3, "Backup Server," includes information about the Backup Server™.
- Chapter 4, "System Database Recovery," includes step-by-step procedures for recovering from various disaster situations. Read this chapter before disasters occur so that recovery will be easier.

- Chapter 5, “Stored Procedure Processing and Management,” describes how SQL Server processes stored procedures. It also provides some of the background information needed to manage stored procedures on your SQL Server.
- Chapter 6, “Encyclopedia of Tasks,”describes a variety of useful tasks, including those required for recovery from disaster situations.
- Chapter 7, “SQL Server Performance Issues,” describes some SQL Server performance issues.
- Chapter 8, “The tempdb Database,” includes information about managing the *tempdb* database.
- Chapter 9, “SQL Server Auditing,” includes information about SQL Server auditing.
- Chapter 10, “Disk Mirroring,” describes disk mirroring issues and what to do in various recovery scenarios.
- Chapter 11, “Novell NetWare Issues,” describes issues commonly encountered when using SQL Server on the Novell NetWare platform.

Your Comments About the Troubleshooting Guide

In order to ensure the continuous improvement of the *SQL Server Troubleshooting Guide*, we need your feedback. To help make it easier for you to provide this feedback, an email alias called “tsg” has been established. The intention of this alias is to allow both Sybase customers and employees to provide comments about the *Troubleshooting Guide* via tsg@sybase.com.

These comments might include:

- Corrections
- Requests for specific additions
- Additions (that is, written material)
- Comments about which sections are particularly helpful
- Comments about which sections are not clear
- Any other input you might have

The goal of the *Troubleshooting Guide* is to assist you in the use and support of Sybase products and to make you more self-sufficient in these activities. To accomplish this, **we need your feedback**.

Style Conventions

Wherever possible, the *SQL Server Troubleshooting Guide* uses the style conventions of the various product manuals. This section contains a brief summary of those conventions.

Style Conventions in Text

Commands and script names appear in bold type; for example:

> To change the **isql** command terminator...

Object names appear in italics; for example:

> Use the **installmodel** script to complete the installation of the *model* database.

SQL Syntax Conventions

The conventions for syntax statements in this manual are as follows:

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in bold.
variable	Variables, or words that stand for values that you fill in, are in italics.
{}	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[]	Brackets mean choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you may select only one of the options shown.
,	The comma means you may choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.

SQL syntax statements (displaying the syntax and options for a command) are printed as follows:

sp_dropdevice [*device_name*]

Examples showing the use of Transact-SQL® commands are printed as follows:

```
1> select * from publishers
2> go
```

Examples of output from the computer are printed as follows:

```
pub_id  pub_name                 city          state
------  ---------------------    -----------   -----
0736    New Age Books            Boston        MA
0877    Binnet & Hardley         Washington    DC
1389    Algodata Infosystems     Berkeley      CA

(3 rows affected)
```

If You Need Help

Help with your SYBASE software is available in the form of documentation and Sybase Technical Support. If you have any questions about the procedures contained in this guide, ask the designated person at your site to contact Sybase Technical Support.

1 SQL Server Memory Issues

This chapter explains how SQL Server manages memory and how you can configure your SQL Server to use memory most efficiently.

All examples in this chapter are based on SQL Server 10.0.1.

➤ ***Note***

For memory configuration issues on the Novell platform, see "Novell Memory Issues" on page 11-1.

How Much Memory Is Available for Your SQL Server?

The maximum configurable memory for SQL Server is 2,147,483,647 2K pages (4GB). If your machine does not have this much memory, you are limited to the amount of memory available on your machine.

To calculate the amount of memory available for SQL Server, subtract the memory required for the operating system and for any user processes and daemons that will be running simultaneously from 4GB. Or subtract the required memory from the maximum memory your machine offers. The remainder is the maximum amount of memory you can allocate to SQL Server.

➤ ***Note***

If the amount of SQL Server memory is increased beyond the size of swap space of your operating system, you may encounter problems at boot or run time. Also, you may encounter performance problems if server memory is increased so high that the operating system begins to page or swap.

How SQL Server Uses Memory

The *run_value* column of the **sp_configure memory** parameter output specifies a certain amount of memory. When SQL Server starts, it requests this amount of memory. For example, if the **memory** parameter has a value of 10,000 pages, SQL Server tries to obtain 19.5MB (10,000 * 2048) of memory. If this amount is not available, SQL Server will not start.

➤ ***Note***

SQL Server uses a 2K (2048-byte) page size on all operating systems except Stratus, where a 4K (4096-byte) page size is used. This discussion assumes a 2K page size.

When SQL Server starts, it allocates memory for several elements:

- SQL Server executable code
- Static memory used by SQL Server
- Non-cache data structures
- Miscellaneous overhead (an additional 5 percent)

The remaining memory is devoted to **total cache space**, which is divided between the data and procedure caches based on the value of the **procedure cache** parameter.

Figure 1-1 depicts these memory elements:

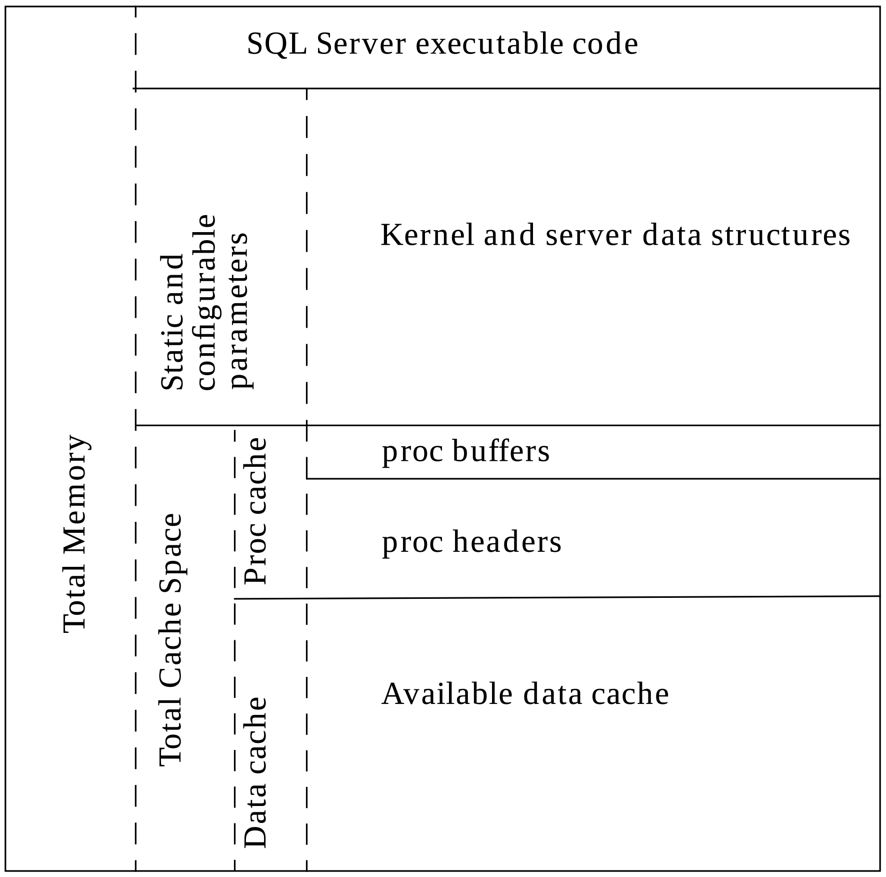
SQL Server executable code
Kernel and server data structures
Static and configurable parameters
proc buffers
proc headers
Proc cache
Total Memory
Total Cache Space
Available data cache
Data cache

Figure 1-1: Elements of memory usage

SQL Server Executable Code

The size of the SQL Server executable code must be subtracted from the total memory available to the SQL Server process. The size of the executable code varies by platform and release, but generally ranges from 2.5MB to 3MB (this size estimate is based on the optimized *dataserver* binary; for *diagserver*, the size is approximately 5MB). Use the **dbcc memusage** command to find the size of your SQL Server's executable. See "Examining Memory Use with dbcc memusage" on page 1-6 for details.

Static Overhead

After the size of the executable has been subtracted, SQL Server allocates an additional amount of static overhead. This amount is not affected by user configurable parameters and normally varies between 1.2MB and 2.25MB.

Configurable Parameters

Next, SQL Server allocates memory for the user-configurable parameters. Table 1-1 shows approximately how much memory is required by some of the parameters that use significant amounts of memory, excluding those parameters related to remote procedure calls.

Table 1-1: Current memory requirements of some user-configurable parameters

Resource	*sp_configure* Value	Bytes per Resource	Space Required (in Megabytes)
User connections	25	23,552 + stack size	1.24[a]
Open databases	12	17,408	0.19
Devices	10	512	0.01
Open objects	500	315	0.15
Locks	5,000	80	0.38
TOTAL			1.97

a. Based on a stack size of 28K. Stack size is configurable, beginning with SQL Server release 4.9.1.

Note that bytes per resource vary by SQL Server release, as shown in Table 1-2.

Table 1-2: Memory requirements for different SQL Server releases

Resource	Bytes per Resource			
	10.0.1 and 10.0.2	**4.9.1**	**4.8**	**Pre-4.8**
User connections	23,552 + stack size	18,000 + stack size	16,000 + stack size	40,961[a]
Open databases	17,408	7,500	6,970	644
Devices	512	512	512	45,056
Open objects	315	315	296	72
Locks	80	70	70	32

a. Based on a stack size of 28K. Stack size is configurable, beginning with SQL Server release 4.9.1

Make sure that you use the appropriate stack size when calculating the amount of memory to allocate to SQL Server. For example, on DEC AXP OSF platforms, the default size is 65K; this means that each user connection requires 88K.

➤ ***Note***

In SQL Server release 10.0.1 or later, if you have changed the **default network packet size** parameter, the calculated value for **user connections** will also change. See “Changing the default network packet size Parameter” on page 1-16 for details.

Total Cache Space

The proportion of total cache space that goes to procedure cache depends on the run value of the **procedure cache** configuration parameter as shown in **sp_configure**. The default value is 20, indicating that 20 percent of the total cache space is used as procedure cache and the remaining 80 percent for data cache.

The amount of SQL Server overhead does not depend on the amount of memory available to it. You can add memory to SQL Server using the **sp_configure memory** parameter. This directly increases the amount of total cache space by the amount of memory added.

◆ ***WARNING!***

Do not increase SQL Server memory beyond available physical memory or the server will start to page fault.

Stored Procedure Memory Requirements

To determine how much memory a stored procedure requires, run **dbcc memusage**. The following fragment of **dbcc memusage** output shows how much memory the stored procedure **ins_comment** requires:

```
Database Id: 5
Object Id: 32003145
Object Name: ins_comment
Version: 1
Uid: 1
Type: stored procedure
Number of trees: 0
Size of trees: 0.000000 Mb, 0.000000 bytes, 0 pages
Number of plans: 1
Size of plans: 0.002758 Mb, 2892.000000 bytes, 2
pages
```

➤ ***Note***

In releases prior to 10.0, **dbcc memusage** only provides information on the first 20 procedures in the cache.

Examining Memory Use

There are two ways to determine how SQL Server is using memory:

- Using the **dbcc memusage** command
- Examining the SQL Server error log

Examining Memory Use with *dbcc memusage*

The **dbcc memusage** command displays information about how SQL Server uses its memory. It provides a snapshot of the data and procedure caches. It is also the only way to determine accurately the size of the SQL Server executable code.

◆ ***WARNING!***

dbcc memusage should be used with care in an SMP environment, as it can generate timeslice errors on a task not running the dbcc command. Timeslice errors that result from a dbcc memusage call occur only on multi-engine servers, and they are more likely to occur on servers that have a large procedure cache.

dbcc memusage does not allow the server to run any other user commands while it is running on a single-processor system.

The larger the amount of SQL Server memory, the longer it will take **dbcc memusage** to traverse the memory chains. SQL Server may appear to hang while **dbcc memusage** is running, because the command puts an exclusive lock on memory while it is running. Therefore, care should be taken when executing this command.

The **dbcc memusage** command must be preceded by the **dbcc traceon** command and followed by the **dbcc traceoff** command, as shown below:

```
1> dbcc traceon(3604)
2> go

1> dbcc memusage
2> go

1> dbcc traceoff(3604)
2> go
```

◆ ***WARNING!***

On SQL Server 4.8, use dbcc traceon in its own batch, or problems may occur.

Following is an example of the Memory Usage section of **dbcc memusage** from a SQL Server with a default configuration:

```
                         Meg  2K Blocks       Bytes
     Configured Memory:  10.0000     5120      10485760

             Code size:   2.7271     1397       2859592
     Kernel Structures:   1.4587      747       1529554
     Server Structures:   1.5777      808       1654320
            Page Cache:   3.3454     1713       3507860
          Proc Buffers:   0.0296       16         31084
          Proc Headers:   0.6973      357        731136

Number of page buffers:     1635
Number of proc buffers:      408
```

- *Code size* indicates the size of the SQL Server executable code.
- *Kernel Structures* and *Server Structures* represent a combination of the static and configurable overhead, as discussed earlier.
- *Page Cache* corresponds to the amount of data cache reserved, including overhead. Overhead includes the hash table that allows pages to be found quickly.
- *Proc Buffers* displays the number of 2K pages allocated to proc buffers. In this example, 16 pages are allocated to proc buffers. This is calculated as follows, rounded up to the nearest page boundary:

  (408 buffers)*(76 bytes / buffer) / (2048 bytes per page) = 16 pages
- *Proc Headers* shows the number of 2K pages allocated to proc headers. In this example, 357 pages are allocated to proc headers. The number of proc headers determines the amount of memory available to store procedures, views, triggers, and other named objects.

For more information on proc buffers and proc headers, see page 1-9.

The size of procedure cache is determined by the size of proc buffers and proc headers combined. In this example, the total size of procedure cache is 373 (357 + 16) pages.

The number of page buffers shows the number of pages that are dedicated to data cache. In this example, 1635 pages (3.19MB) are dedicated to data cache.

As of release 10.0, **dbcc memusage** also prints each procedural object's cache site.

Determining Total Cache Space from the SQL Server Error Log

Memory-related messages in the SQL Server error log state exactly how much data and procedure cache is allocated, and how many procedures or other compiled objects can reside in cache at any one time. These messages provide the most accurate information regarding cache usage on SQL Server.

On a development system, you may want to increase the amount of memory dedicated to the procedure cache. On a production system, however, you may want to reduce the size of procedure cache again in order to obtain more memory for the data cache. Refer to "Procedure Cache Sizing" on page 1-11 for recommendations on optimizing procedure cache size.

A SQL Server with a default configuration (that is, **memory** parameter = 5120 pages) would yield the following memory-related error log messages:

```
server: Number of buffers in buffer cache: 1635.
server: Number of proc buffers allocated: 408.
server: Number of blocks left for proc headers:357.
```

Each of these error log messages is described below.

Data Cache Messages

```
server: Number of buffers in buffer cache: 1635.
```

This message indicates how many page buffers are allocated to the data cache. In this example, 1635 pages are dedicated to the data cache.

Procedure Cache Messages

```
server: Number of proc buffers allocated: 408
```

This message states the total number of proc buffers allocated in the procedure cache.

A **proc buffer** (procedure buffer) is a data structure used to manage compiled objects (any stored procedure, trigger, rule, default, check constraint, or view) in the procedure cache. One proc buffer is used for every copy of a named object stored in the procedure cache. When SQL Server starts, it determines the number of proc buffers required and multiplies that value by the size of a single proc buffer (76 bytes) to obtain the total amount of memory required. It then allocates this amount of memory, treated as an array of proc buffers. Unlike some data structures, proc buffers can span pages.

This message indicates the total number of **proc headers** available for use in the procedure cache:

```
server: Number of blocks left for proc headers:357.
```

proc headers (procedure headers) are where compiled objects (such as stored procedures) are stored. Depending on the size of the object to be stored, one or more proc headers may be required. The total number of compiled objects that can be stored in the procedure cache is limited by the number of available proc headers or proc buffers, whichever is less. In this example, no more than 357 compiled objects are allowed. Because stored procedures often use up more than one page, the practical value of this number may be even lower.

The total size of procedure cache is the combined total of memory allocated to proc buffers (rounded up to the nearest page boundary) plus the memory allocated to proc headers. In this example, 408 proc buffers are allocated; thus, 16 pages are used for proc buffers:

(408 buffers) * (76 bytes / buffer) / (2048 bytes / page) = 16 pages

Adding these 16 pages to the 357 pages allocated for proc headers, the total amount of memory reserved for procedure cache is 373 2K pages.

Tuning Memory Use

This section provides background information and techniques for tuning SQL Server’s memory use.

Manipulating Cache

There are several issues related to procedure and data cache that you should consider when tuning your SQL Server.

Procedure Cache vs. Data Cache

When the **memory** configuration parameter is changed, the amount of memory dedicated to the data and procedure caches also changes.

Similarly, when the value of the **procedure cache** parameter is changed, memory is shifted between the two caches, so each cache has a different size than in the previous configuration. There is no one-step method to increase or decrease only the data cache or only the procedure cache.

This makes it harder to determine how much data and procedure cache is best for a particular SQL Server. It is best to change one thing at a time to more accurately measure the effect of the change. The **memory** and **procedure cache** parameters can be used together effectively to increase only one of the caches. The following example shows how to do this.

Consider a SQL Server configured with 10MB (5120 pages) of memory, 50 user connections, and a 28K stack size. The error log shows that 3.2MB (1638 pages) is dedicated to the data cache and 0.7MB (358 pages) is dedicated to the procedure cache. Therefore, the total cache space on this server is 3.9MB (1996 pages).

For this example, assume that the stored procedures on this SQL Server need 440 pages of procedure cache to store one copy of each of the main procedures in cache at one time.

To accomplish this without significantly changing the amount of data cache, you must increase both the **memory** and the **procedure cache** parameters. The new value for **memory** is simply the old value plus the number of additional pages needed:

memory = 5,120 + (440 - 358) = 5202 pages

Because adding memory does not increase the amount of overhead, the total cache space increases to (1,638+440-358) = 1720 pages.

You can determine the value for the **procedure cache** parameter as follows:

procedure cache = (Total Cache Space) * (Procedure Cache%)

Therefore:

procedure cache % = Desired Procedure Cache / Total Cache Space

So, for this example:

procedure cache % = 440 pages / 1720 pages = 26%

Note that this calculation can be used to increase the data cache and leave the procedure cache constant.

Procedure Cache Sizing

You can estimate how much procedure cache is needed for best SQL Server performance by estimating how much is needed to optimize the procedure cache hit rate.

The following calculation is a good starting point for sizing the procedure cache:

Procedure Cache Size = Max(# of concurrent users) * (size of largest plan) * 1. 25

The following example describes one way to size your procedure cache.

Consider a SQL Server application environment consisting of 500 stored procedures. Of these, 100 are used frequently throughout the day and the other 400 are executed only periodically. To avoid the run-time overhead of regenerating query plans and loading them into cache, it is desirable for the procedure cache to be sized so that at least one copy of every main procedure can fit in cache.

To accomplish this, you must determine the amount of memory needed to store the query plans of the main procedures using **dbcc memusage**. For instructions, see “Examining Memory Use with dbcc memusage” on page 1-6. You can then estimate the total size needed by examining a few representative procedures and determining an average plan size. Remember that plans use 2K buffers in the procedure cache, so the smallest a plan can be is 2K (or one page).

Use the following formula to estimate the minimum procedure cache size:

Minimum Procedure Cache Needed = (# of main procedures) * (avg. plan size)

The minimum procedure cache size is the smallest amount of memory that allows at least one copy of each frequently used compiled object to reside in cache.

For this example, assume that the main procedures have an average plan size of 15 pages. To allow at least one copy of each of these procedures to reside in cache, you need:

Minimum Procedure Cache Needed = (100 procs) * (15 pgs / proc) = 1500 pages

Once you determine the minimum procedure cache value, add an additional amount to allow for the concurrent use of the main procedures, as well as the lesser-used procedures. This amount varies, but an extra 10 percent is a good starting point.

The preceding methods are just two approaches to sizing the procedure cache. Depending on the requirements of your specific SQL Server environment, you may be able to achieve sufficient performance using a smaller procedure cache than indicated by this calculation. If memory resources are limited, run enough tests to determine exactly how much procedure cache is best for your environment.

Running Out of Procedure Cache

The procedure cache is finite and various limitations may be encountered. Two such limitations are described below.

- There is insufficient available procedure cache to load in another query tree or plan.

  In SQL Server releases 10.0 and later, there is no limitation on the number of pages for query plans and trees. The 64-page limit, which causes SQL Server to raise Error 703, is applicable to 4.9.x and earlier releases of SQL Server. However, in all releases, query plans and trees will grow. See “Error 703” in *SQL Server Error Messages* for additional details.

In SQL Server releases 10.0 and later, you can prevent query trees from growing automatically, start the SQL Server with trace flag 241. Refer to “How to Start SQL Server with Trace Flags” on page 6-18 for information about trace flags.

◆ ***WARNING!***

In System 10, if you use trace flag 241, each time a stored procedure is normalized, SQL Server will restore the procedure to its original size. This may be resource-intensive.

Note that you can free up space in the procedure cache only if a tree or plan is not currently in use.

For more information on query plans and trees, see Chapter 5, “Stored Procedure Processing and Management.”

- The maximum number of procedures, or other compiled objects, has been reached. It is possible for a procedure execution or creation to fail even when there is adequate space in cache to hold the resulting plan. This will occur if the limit of concurrent objects in procedure cache has already been reached. The number of objects allowed in procedure cache is a function of the procedure cache size procedure buffers and is recorded in the error log during start-up.

  See “Determining Total Cache Space from the SQL Server Error Log” on page 1-8 for details. If the maximum number of compiled objects is reached, SQL Server will also display Error 701. See “Error 701” in *SQL Server Error Messages* for additional details.

Data Cache Size Restrictions on Pre-4.8 SQL Server

Early releases of SQL Server had some restrictions on the amount of data cache supported. There are no negative ramifications associated with increasing data or procedure cache in SQL Server releases 4.8 and later.

SQL Server releases prior to 4.8 had a definite threshold value for how much data cache they could support. The reason for this was that these releases had a set number (1024) of hash buckets, which were used to quickly find out whether a referenced page was already in cache. As the size of the data cache grew, the length of the hash chains also grew, since there were only a limited number of buckets allocated. Because of this, after a certain point, adding data cache actually decreased performance.

Even if you are running a pre-4.8 SQL Server, there is no single maximum data cache value that applies to every site. Some sites gain performance by going well over 30MB; others have to use less.

For releases 4.8 and later, a dynamic number of hash buckets is available. It is unlikely that a hash chain will ever grow to more than 2 pages long. Therefore, increasing data cache on SQL Server release 4.8 or later does not decrease performance.

On very large configurations, it is still possible to experience some degradation in performance. For example, a 4GB cache would have typical hash chain sizes between 8 and 16 pages long.

User Connections

This section discusses tuning issues related to procedure and data cache.

Increasing User Connections

Each configured user connection requires a significant amount of memory, **whether or not the connection is used**. This amount depends mostly on the run value of the **stack size** parameter, as shown with **sp_configure**. (The **stack size** parameter became user-configurable in SQL Server 4.9.1.)

The overhead required for each user connection can be calculated as follows:

Overhead per User Connection	=	21,504 bytes + stack size + 2K (2K is the stack guard area size)

➤ ***Note***

The calculated value will be different if you changed the value of the **default network packet size** parameter. Refer to “Changing the default network packet size Parameter” on page 1-16 for details.

Starting with SQL Server 10.0.1, the default **stack size** value on most platforms is 28,672 (28K) plus 2048 (2K, the stack guard size area). Therefore, for a SQL Server running with the default **stack size**, there is about 51K of overhead per user connection. See "Memory requirements for different SQL Server releases" on page 1-3 for more information about SQL Server resources and memory needs.

When you configure additional user connections and restart SQL Server, the additional overhead reduces the data and procedure cache sizes for the SQL Server. Do not configure more user connections than necessary. Also, anytime you increase user connections, you may also have to increase the **memory** parameter to give SQL Server additional memory for configuring the connections. Make sure that you have adequate physical memory before increasing SQL Server memory. Otherwise, you may have to decrease some other configured parameter. See "How SQL Server Uses Memory" on page 1-1.

➤ ***Note***

If you use the VMS operating system, do not forget to recalculate the quota file for SQL Server whenever you change either the **user connections** or **memory** parameters.

User Connections and Your Operating System

Remember that the operating system, not SQL Server, is the limiting factor for user connections. **sp_configure** only increases the number of user connections that SQL Server allows; it does not affect the number of user connections the underlying operating system can support. For example, if the operating system allows only 64 file descriptors per process, increasing the user connections beyond 64 in SQL Server will have no effect and could cause problems.

User Connections and SQL Server Memory Resources

Each configured user connection requires a significant amount of memory. If you use **sp_configure** to increase the value of **user connections**, the additional memory is required as soon as the server is rebooted.

Changing the *default network packet size* Parameter

The **default network packet size** parameter is configurable starting with SQL Server 10.0. Each user connection uses three network buffers, each requiring **default network packet size** bytes. If the **default network packet size** parameter is changed, then the memory overhead required for each user connection also changes.

The total amount of memory allocated for network packets is:

> **user connections** * 3 * **default network packet size**

The default value for the **default network packet size** parameter is 512 bytes; therefore, 25 * 3 * 512 = 38,400 bytes are allocated for a system with 25 user connections. If you increase **default network packet size** from 512 to 1024 bytes, 25 * 3 * 1024 = 76,800 bytes of memory will be allocated for user connections, decreasing available cache by 38,400 bytes.

If you increase the **default network packet size**, check the **memory** configuration and **user connections** variables to make sure that you leave enough space for other SQL Server memory needs, especially data and procedure cache. **dbcc memusage** will **not** reflect the increase in memory use by SQL Server. On UNIX platforms, you can use **ipcs -mb** to see that memory is added to the kernel structures.

User Connections and the Threshold Manager

Thresholds were introduced in SQL Server release 10.0 to monitor available space in a database. When a threshold is activated, it accesses SQL Server as a **detached user**, which uses only one user connection per threshold. It is imperative that you allow enough user connections to enable thresholds to attach to SQL Server when necessary.

The *user connections* Parameter and *master..spt_values*

➤ ***Note***

The material in this section does **not** apply to System 10™ or later releases.

◆ ***WARNING!***

Read this entire section before altering the user connections parameter.

Sometimes, you must increase the number of user connections allowed by SQL Server. However, you cannot use **sp_configure** to specify a configuration parameter that is higher than the current maximum value for that parameter. The rest of this section provides information about the effects of changing this parameter and provides instructions for increasing user connections from the current maximum value.

Increasing the maximum value of **user connections** does not require overhead. The memory is only required when the *run_value* column of **sp_configure** output is changed (when you increase the value of **user connections**).

An internal SQL Server table called *master..spt_values* stores the name, minimum, and maximum values of all the parameters listed in **sp_configure**. The SYBASE System Administrator can update this table to change the minimum or maximum values for a particular parameter.

The format of the *master..spt_values* table is shown below.

Table 1-3: Format of the table master..spt_values

Column Name	Datatype	Length	Nulls
name	*varchar*	20	1
number	*int*	4	0
type	*char*	1	0
low	*int*	4	1
high	*int*	4	1

◆ ***WARNING!***

The *spt_values* table also stores various other information used by SQL Server. Be extremely careful when you modify it. Do not change any entries in *spt_values* except the user connections entry unless specifically directed to do so by Sybase Technical Support.

How to Safely Change the *user connections* Parameter

➤ ***Note***

The material in this section does **not** apply to System 10 or later releases.

Execute the following steps to safely change the **user connections** parameter:

1. Run this query to see how many connections the operating system supports:

   ```
   1> select @@max_connections
   2> go
   ```

   Do not set **user connections** above this value.

◆ ***WARNING!***

On STREAMS-based operating systems, you may run out of STREAMS resources if you reset the *spt_values* as shown in this procedure. You may want to contact Sybase Technical Support before using this procedure.

2. Execute the following commands to prepare to change the parameter value:

   ```
   1> sp_configure allow,1
   2> go
   
   1> reconfigure with override
   2> go
   
   1> use master
   2> go
   
   1> begin tran
   2> go
   
   1> update spt_values set high = new_parameter_value
   2> where number = 103 and type = "C"
   3> go
   ```

   where *new_parameter_value* is the value you want **user connections** to have. Do not insert a number higher than the number you obtained in step 1!

3. Check the output of the **update** command. If it affected more than one row, issue a **rollback transaction** command. If only one row was affected, execute **sp_configure**, without any arguments, to check the new maximum value for user connections. If it is correct, commit the transaction and disable updates to the system catalog:

   ```
   1> commit tran
   2> go
   ```

```
1> sp_configure allow,0
2> go
1> reconfigure with override
2> go
```

4. You can increase the value of the **user connections** parameter, up to the maximum, using the **sp_configure** system procedure as described in the *SQL Server Reference Manual*.

Multiple-Engine Issues

This section discusses tuning issues specific to SQL servers running on multi-processor machines.

CPU Time and the *cschedspins* Parameter

SQL Server release 4.8 and later releases uses CPU time even when idle, because of a new network polling scheme used to provide improved response time. Previous releases of SQL Server used a virtual timer. This timer functioned only when the process was running. However it was not adequate for SQL Server, which needs to look continuously for new connections.

To provide better performance no matter how many users are connected to SQL Server, the virtual timer has been replaced with a real timer. This delivers signals whether the process is idle, waiting for disk I/O, or running. Since network polling is constant, response to user requests is faster.

The real timer uses more CPU time because when signals are delivered while SQL Server is idle, the appropriate clock interrupt handler in SQL Server activates, causing the underlying operating system kernel to add time to the SQL Server process. Therefore, users will see a uniform increase in the amount of time consumed by an idle SQL Server. This is normal. However, if SQL Server operation itself seems slow, for example if bulk copy (**bcp**) is too slow, you may be able to increase the speed by adjusting the **cschedspins** configuration parameter.

The value of **cschedspins** can be changed only if you have SQL Server release 4.8 or later. Previous releases do not have this parameter.

◆ ***WARNING!***

Exercise caution when changing the cschedspins parameter. Contact Sybase Technical Support for details.

For SQL Server running a single engine, the optimal value is 1 for all releases. The optimal value for multiple engines is usually 32 for release 4.8 and 2000 (the default value) for releases 4.9, 4.9.1., and 10.x.

To change the value of **cschedspins** follow these steps:

1. Shut down SQL Server.
2. Change the value of **cschedspins** using the appropriate command from the following table:

Table 1-4: Adjusting cschedspins commands

Platform	Command
UNIX	**buildmaster -d*DEVICE_NAME* -ycschedspins=*new_value***
VMS	**BUILDMASTER/DISK=*DEVICE_NAME*/ALTER="cschedspins"=*new_value***
Novell NetWare	**load bldmastr -d *device_name* -ycschedspins=*new_value***
OS/2	**bldmastr -d *device_name* -ycschedspins=*new_value***
Windows NT	**bldmastr -d *device_name* -ycschedspins=*new_value***
SCO UNIX	**buildmaster -d *device_name* -ycschedspins=*new_value***

   DEVICE_NAME is the full path and name of the master device as it appears in the runserver file (except Novell NetWare, which does not use a runserver file).

   If changing the value to the recommended setting in Table 1-4 does not improve performance, try other values in increments of 8. The value of **cschedspins** should always be 1 or a multiple of 8. Be sure to use the current version of **buildmaster** when making this change.

3. Restart SQL Server.

➤ ***Note***

Reset the **cschedspins** parameter anytime you rebuild the master device.

Using *cclkrate* Parameter (SQL Server 4.9.1 and Later)

The parameter **cclkrate** (clock rate) measures time in microseconds. The value of **cclkrate** controls the length of time for each SQL Server **tick**. A tick is the lowest value possible for the parameter **time slice** (adjustable via **sp_configure**). Ticks control the following functions:

- Housekeeping functions are performed at each tick (or 10 times a second if **cclkrate** = 100,000 or 200,000 for VMS).
- When a UNIX SQL Server is busy, it checks the network only once per tick. When idle, SQL Server checks more frequently. This increases throughput during busy times while keeping response rates fast when SQL Server is less active.

Increasing the value of **cclkrate** increases the response time when SQL Server is busy. It allows CPU-intensive tasks to dominate SQL Server use. Decreasing the value of **cclkrate** reduces performance because SQL Server performs its housekeeping functions more often.

◆ ***WARNING!***

Changing the value of cclkrate has ramifications that are far-reaching and not always obvious. It is recommended that you do not change the value unless you are aware of all the ramifications of this change. Be sure to carefully test your SQL Server after changing the value of cclkrate.

To change the value of **cclkrate**, follow these steps:

1. Shut down SQL Server.
2. Change the value of **cclkrate** using the appropriate command:

Table 1-5: Adjusting cclkrate commands

Platform	Command
UNIX	**buildmaster -d*DEVICE_NAME* -ycclkrate=*new_value***
VMS	**BUILDMASTER/DISK=*DEVICE_NAME*/ALTER="cclkrate"=*new_value***
Novell NetWare	**load bldmastr -d *DEVICE_NAME*-ycclkrate=*new_value***
OS/2	**bldmastr -d *device_name* -ycclkrate=*new_value***

Table 1-5: Adjusting cclkrate commands

Platform	Command
Windows NT	**bldmastr -d** ***device_name*** **-ycclkrate=*new_value***
SCO UNIX	**buildmaster -d** ***device_name*** **-ycclkrate=*new_value***

DEVICE_NAME is the full path and name of the master device as it appears in the runserver file.

3. Restart SQL Server.

Reset the **cclkrate** parameter again anytime you rebuild the master device.

Setting *max online engines parameter* in *sp_configure*

Beginning with SQL Server release 4.8, a configuration parameter exists for setting the number of engines in a multiprocessor SQL Server: **max online engines**. This parameter indicates the number of engines SQL Server will use.

Two related parameters appear in **sp_configure** output, **min online engines** and **engine adjust interval**:

- **min online engines** cannot exceed the number of CPUs available on your system, and must be set to at least 1.
- **engine adjust interval** is not currently used, but may be available in a future release.

2 Transaction Log Management

The purpose of this chapter is to assist you in your long term management of transaction logs, including sizing and truncating the transaction log, managing large transactions, and bulk copy. An advanced section discusses automating transaction log dumps and detecting long-running transactions.

About Transaction Logs

Most SQL Server processing is logged in the transaction log table, *syslogs*. Each database, including the system databases *master, model, sybsystemprocs*, and *tempdb*, has its own transaction log. As modifications to a database are logged, the transaction log continues to grow until it is truncated, either by a **dump transaction** command or automatically if the **trunc log on chkpt** option is turned on as described below. This option is not recommended in most production environments where transaction logs are needed for media failure recovery, because it does not save the information contained in the log.

The transaction log on SQL Server is a write-ahead log. After a transaction is committed, the log records for that transaction are guaranteed to have been written to disk. Changes to data pages may have been made in data cache but may not yet be reflected on disk.

◆ ***WARNING!***

This guarantee cannot be made when UNIX files are used as SYBASE devices.

Transaction Logs and *commit transaction*

When you issue a **commit transaction**, the transaction log pages are immediately written to disk to ensure recoverability of the transaction. The modified data pages in cache might not be written to disk until a checkpoint is issued by a user or SQL Server or periodically as the data cache buffer is needed by other SQL Server users. Note that pages modified in data cache can be written to disk prior to the transaction committing, but not before the corresponding log records have been written to disk. This happens if buffers in data cache containing dirty pages are needed to load in a new page.

Transaction Logs and the *checkpoint* Process

If the **trunc log on chkpt** option is set for a database, SQL Server truncates the transaction log for the database up to the page containing the oldest outstanding transaction when it issues a checkpoint in that database. A transaction is considered outstanding if it has not yet been committed or rolled back. A **checkpoint** command issued by a user does not cause truncation of the transaction log, even when the **trunc log on chkpt** option is set. Only implicit **checkpoint**s performed automatically by SQL Server result in this truncation. These automatic **checkpoint**s are performed using the internal SQL Server process called the **checkpoint** process.

The **checkpoint** process wakes up about every 60 seconds and cycles through every database to determine if it needs to perform a **checkpoint.** This determination is based on the **recovery interval** configuration parameter and the number of rows added to the log since the last **checkpoint.** Only those rows associated with committed transactions are considered in this calculation.

If the **trunc log on chkpt** option is set, the **checkpoint** process attempts to truncate the log every sixty seconds, regardless of the recovery interval or the number of log records. If nothing will be gained from this truncation, it is not done.

Transaction Logs and the *recovery interval*

The **recovery interval** is a configuration parameter that defines the amount of time for the recovery of a single database. If the activity in the database is such that recovery would take longer than the **recovery interval**, the SQL Server checkpoint process issues a checkpoint. Because the checkpoint process only examines a particular database every 60 seconds, enough logged activity can occur during this interval that the actual recovery time required exceeds the time specified in the **recovery interval** parameter.

Note that the transaction log of the *tempdb* database is automatically truncated during every cycle of the checkpoint process, or about every 60 seconds. This occurs whether the **trunc log on chkpt** option is set on *tempdb* or not.

Turning Off Transaction Logging

Transaction logging performed by SQL Server cannot be turned off, to ensure the recoverability of all transactions performed on SQL Server. Any SQL statement or set of statements that modifies data is a transaction and is logged. You can, however, limit the amount of logging performed for some specific operations, such as bulk copying data into a database using bulk copy (**bcp**) in the fast mode, performing a **select/into** query, or truncating the log. See the *Tools and Connectivity Troubleshooting Guide* and the *SQL Server Reference Manual* for more information on **bcp**. These minimally logged operations cause the transaction log to get out of sync with the data in a database, which makes the transaction log useless for media recovery.

Once a non-logged operation has been performed, the transaction log cannot be dumped to a device, but it can still be truncated. You must do a **dump database** to create a new point of synchronization between the database and the transaction log to allow the log to be dumped to device.

What Information Is Logged

When a transaction is committed, SQL Server logs every piece of information relating to the transaction in the transaction log to ensure its recoverability. The amount of data logged for a single transaction depends on the number of indexes affected, the amount of data changed, and the number of pages that must be allocated or deallocated. Certain other page management information may also be logged. For example, when a single row is updated, the following types of records may be placed in the transaction log:

- A data delete record, including all the data in the original row.
- A data insert record, including all the data in the modified row.
- One index delete record per index affected by the change.
- One index insert record per index affected by the change.
- One page allocation record per new data/index page required.
- One page deallocation record per data/index page freed.

Sizing the Transaction Log

There is no hard and fast rule dictating how big a transaction log should be. For new databases, a log size of about 20 percent of the overall database size is a good starting point. The actual size required depends on how the database is being used; for example:

- The rate of **update**, **insert**, and **delete** transactions
- The amount of data modified per transaction
- The value of the **recovery interval** configuration parameter
- Whether or not the transaction log is being saved for media recovery purposes

Because there are many factors involved in transaction logging, you usually cannot accurately determine in advance how much log space a particular database requires. The best way to estimate this size is to simulate the production environment as closely as possible in a test. This includes running the applications with the same number of users as will be using the database in production.

Separating Data and Log Segments

Always store transaction logs on a separate database device and segment from the actual data. If the data and log are on the same segment, you cannot save transaction log dumps. Up-to-date recovery after a media failure is therefore not possible. If the device is mirrored, however, you may be able to recover from a hardware failure. Refer to the *System Administration Guide* for more information.

Also, the data and log segments must be on separate segments so that you can determine the amount of log space used. **dbcc checktable** on *syslogs* only reports the amount of log space used and what percentage of the log is full if the log is on its own segment.

Finally, because the transaction log is appended each time the database is modified, it is accessed frequently. You can increase performance for logged operations by placing the log and data segments on different physical devices, such as different disks and controllers. This divides the I/O requests for a database between two devices.

Truncating the Transaction Log

The transaction log must be truncated periodically to prevent it from filling up. You can do this either by enabling the **trunc log on chkpt** option or by regularly executing the **dump transaction** command.

◆ ***WARNING!***

Up-to-the-minute recoverability is not guaranteed on systems when the trunc log on chkpt option is used. If you use this on production systems and a problem occurs, you will only be able to recover up to your last database dump.

Because the **trunc log on chkpt** option causes the equivalent of the **dump transaction with truncate_only** command to be executed, it truncates the log without saving it to a device. Use this option only on databases for which transaction log dumps are not being saved to recover from a media failure, usually only development systems.

Even if this option is enabled, you might have to execute explicit **dump transaction** commands to prevent the log from filling during peak loads.

If you are in a production environment and using **dump transaction** to truncate the log, space the commands so that no process ever receives an 1105 (out of log space) error.

When you execute a **dump transaction**, transactions completed prior to the oldest outstanding transaction are truncated from the log, unless they are on the same log page as the last outstanding transaction. All transactions since the earliest outstanding transaction are considered active, even if they have completed, and are not truncated. Figure 2-1 illustrates active and outstanding transactions.

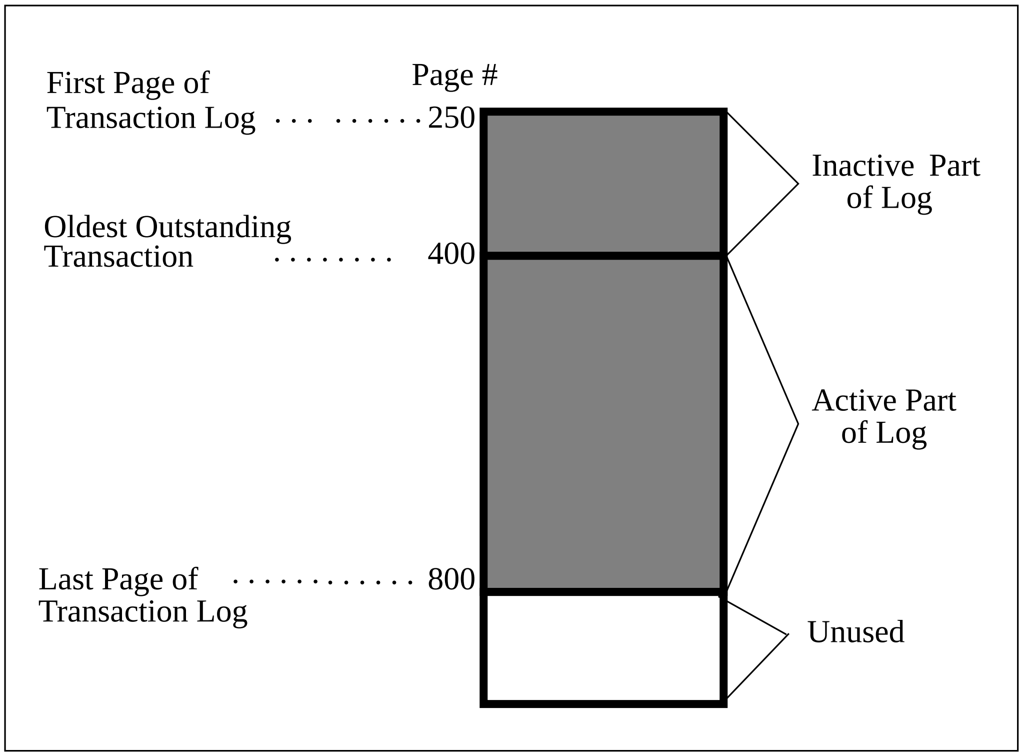
First Page of Transaction Log
Page #
250
Oldest Outstanding Transaction
400
Last Page of Transaction Log
800
Inactive Part of Log
Active Part of Log
Unused

Figure 2-1: Active transactions and outstanding transactions illustrated

This figure shows that all transactions after an outstanding transaction are considered active. Note that the page numbers do not necessarily increase over time.

Because the **dump transaction** command only truncates the inactive portion of the log, you should not allow stranded transactions to exist for a long time. For example, suppose a user issues a **begin transaction** command and never commits the transaction. Nothing logged after the **begin transaction** can be purged out of the log until one of the following occurs:

- The user issuing the transaction completes it.
- The user process issuing the command is forcibly stopped, and the transaction is rolled back.
- SQL Server is shut down and restarted.

Stranded transactions are usually due to application problems but can also occur as a result of operating system or SQL Server errors. See “Managing Large Transactions” on page 2-6 and “Long-Running Transactions” on page 2-10 for more information.

Managing Large Transactions

Because of the amount of data SQL Server logs, it is important to manage large transactions efficiently. Four common transaction types can result in extensive logging:

- Mass updates
- Deleting a table
- Insert based on a subquery
- Bulk copying in

The following sections contain explanations of how to use these transactions so that they do not cause extensive logging.

Mass Updates

The following SQL statement updates every row in the *large_tab* table. All of these individual updates are part of the same single transaction.

```
1> update large_tab set col_1 = 0
2> go
```

On a large table, this query results in extensive logging, often filling up the transaction log before completing. In this case, an 1105 error (transaction log full) results. The portion of the transaction that was processed is rolled back, which can also require significant server resources.

Another disadvantage of unnecessarily large transactions is the number or type of locks held. An exclusive table lock is normally acquired for a mass update, which prevents all other users from modifying the table during the update. This may cause deadlocks.

You can sometimes avoid this situation by breaking up large transactions into several smaller ones and executing a **dump transaction** between the different parts. For example, the single update statement above could be broken into two or more pieces as follows:

```
1> update large_tab set col1 = 0
2> where col2 < x
3> go

1> dump transaction database_name
2> with truncate_only
3> go
```

```
1> update large_tab set col1 = 0
2> where col2 >= x
3> go

1> dump transaction database_name
2> with truncate only
3> go
```

This example assumes that about half the rows in the table meet the condition *col2 < x* and the remaining rows meet the condition *col2 >= x*.

If transaction logs are saved for media failure recovery, the log should be dumped to a device and the **with truncate_only** option should not be used. Once you execute a **dump transaction with truncate_only**, you must dump the database before you can dump the transaction log to a device.

Delete Table

The following SQL statement deletes the contents of the *large_tab* table within a single transaction and logs the complete before-image of every row in the transaction log:

```
1> delete table large_tab
2> go
```

If this transaction fails before completing, SQL Server can roll back the transaction and leave the table as it was before the **delete**. Usually, however, you do not need to provide for the recovery of a **delete table** operation. If the operation fails halfway through, you can simply repeat it and the result is the same. Therefore, the logging done by an unqualified **delete table** statement may not always be needed.

You can use the **truncate table** command to accomplish the same thing without the extensive logging:

```
1> truncate table large_tab
2> go
```

This command also deletes the contents of the table, but it logs only space deallocation operations, not the complete before-image of every row.

Insert Based on a Subquery

The SQL statement below reads every row in the *large_tab* table and inserts the value of columns *col1* and *col2* into *new_tab*, all within a single transaction:

```
1> insert new_tab select col1, col2 from large_tab
2> go
```

Each insert operation is logged, and the records remain in the transaction log until the entire statement has completed. Also, any locks required to process the inserts remain in place until the transaction is committed or rolled back. This type of operation may fill the transaction log or result in deadlock problems if other queries are attempting to access *new_tab*. Again, you can often solve the problem by breaking up the statement into several statements that accomplish the same logical task. For example:

```
1> insert new_tab
2> select col1, col2 from large_tab where col1 <= y
3> go

1> dump transaction database_name
2> with truncate_only
3> go

1> insert new_tab
2> select col1, col2 from large_tab where col1 > y
3> go

1> dump transaction database_name
2> with truncate_only
3> go
```

➤ ***Note***

This is just one example of several possible ways to break up a query.

This approach assumes that *y* represents a median value for *col1*. It also assumes that null values are not allowed in *col1*. The inserts run significantly faster if a clustered index exists on *large_tab.col1*, although it is not required.

If transaction logs are saved for media failure recovery, the log should be dumped to a device and the **with truncate_only** option should not be used. Once you execute a **dump transaction with truncate_only**, you must dump the database before you can dump the transaction log to a device.

Bulk Copy

You can break up large transactions when using **bcp** to bulk copy data into a database. If you use **bcp** without specifying a batch size, the entire operation is performed as a single logical transaction. Even if another user process does a **dump transaction** command, the log records associated with the bulk copy operation remain in the log until the entire operation completes and another **dump transaction** command is performed. This is one of the most common causes of the 1105 error. You can avoid it by breaking up the bulk copy operation into batches. Use this procedure to ensure recoverability:

1. Turn on the **trunc log on chkpt** option:

```
1> use master
2> go

1> sp_dboption database_name,
2> trunc, true
3> go

1> use database_name
2> go

1> checkpoint
2> go
```

➤ ***Note***

"trunc" is an abbreviated version of the option **trunc log on chkpt**.

2. Specify the **batch size** on the **bcp** command line. This example copies rows into the *pubs2.authors* table in batches of 100:

Platform	Command
UNIX	**bcp ... -b 100**
Digital OpenVMS	**bcp ... /batch_size=100**
Novell NetWare	**load bcp pubs2.authors in phonebook -b 100**
OS/2	**bcp pubs2.authors in phonebook -b 100**
Windows NT	**bcp pubs2.authors in phonebook -b 100**
SCO UNIX	**bcp pubs2.authors in phonebook -b 100**

3. Turn off the **trunc log on chkpt** option when the **bcp** operations are complete, and dump the database.

In this example, a batch size of 100 rows is specified, resulting in one transaction per 100 rows copied. You may also need to break the **bcp** input file into two or more separate files and execute a **dump transaction** between the copying of each file to prevent the transaction log from filling up.

If the **bcp in** operation is performed in the fast mode (with no indexes or triggers), the operation is not logged. In other words, only the space allocations are logged, not the complete table. The transaction log cannot be dumped to a device in this case until after a database dump is performed (for recoverability).

If your log is too small to accommodate the amount of data being copied in, you may want to do batching and have the **sp_dboption trunc log on checkpoint** set. This will truncate the log after each checkpoint.

See the *Tools and Connectivity Troubleshooting Guide* and the *SQL Server Reference Manual* for more information on **bcp**.

Long-Running Transactions

A single, long-running transaction can prevent the log from being truncated. This occurs because SQL Server only dumps the inactive portion of a transaction log.These transactions are often a result of application errors or badly formed queries but they can also result from operating system or SQL Server problems. It is important to detect the presence of these transactions and act accordingly. Otherwise, the transaction log eventually fills up, even if **dump transaction** commands are executed.

Causes of Long-Running Transactions

Some of the causes for a long-running transaction include:

- An incorrectly written query that runs for many hours. Queries that create cartesian products or include user input are common mistakes in query writing.
- An application error that starts a transaction but never completes it.
- Operating system error.
- Network error.
- SQL Server error.
- Hardware problems.

How to Detect Long-Running Transactions

To query the system tables in the database to find a long-running transaction, follow these steps:

1. Run the following query:

   ```
   1> select first from sysindexes
   2> where id = 8
   3> go
   ```

   Note the page number in the output. That indicates the first page of the *syslogs* table (*syslogs* ID = 8).

2. Do a **dump transaction** command.

3. Rerun the query from step 1, and note the page number of *syslogs* again. If the first page of the log is the same, either the log is only one page long or an outstanding transaction is preventing the log from being purged.

Once this condition has been detected, your knowledge of the applications using the database can tell you whether or not it represents a problem. For example, suppose the longest running transaction on a database takes an average of 2 minutes and the log could not be purged over a 5-minute interval. The average of 2 minutes could have been exceeded due to machine or SQL Server load, and there is a good chance that no problem exists.

How to Clear Long-Running Transactions

You can clear a long-running transaction by restarting SQL Server. This causes the database to go through normal recovery, so any outstanding transactions are either committed or rolled back. If the long-running transaction is due to a runaway query, and the transaction has been identified, use the **kill** command to stop the process. This clears the transaction and allows the log to be truncated. If the **kill** command cannot stop the process, you must restart SQL Server to resolve the problem.

If this type of problem occurs frequently, Sybase Technical Support may be able to identify which process is involved.

Automating Transaction Log Dumps

You can create an application that automatically dumps the transaction log to a device, to help prevent the transaction log from filling up.

An application that dumps the transaction log and saves the dumped portions for recovery should:

- Periodically check the transaction log to determine if it needs to be dumped
- Dump the transaction log to a device

Although it is possible to simply dump the log without saving the dumped portion, this is not recommended because it renders the SQL Server less than completely recoverable in the event of failure. See the *SQL Server Reference Manual* entry for **dump transaction** for more information.

How to Check Transaction Log Size

The following sections present some methods for checking the transaction log size, depending on your SQL Server release level.

SQL Server Release Prior to 4.8

Output from **dbcc checktable** on *syslogs* includes messages similar to the following, starting with SQL Server release 4.0.1:

```
*** NOTICE: Space used on the log segment is 12.87 Mbytes, 64.35%
*** NOTICE: Space free on the log segment is  7.13 Mbytes, 35.65%
```

You can create an operating system script file which executes **dbcc checktable** on *syslogs* periodically, extracts the needed usage information on the log, and determines if you need a **dump transaction** command.

SQL Server Release 4.8 and Later

There are two other ways you can determine how full the transaction log is. These are:

- Execute the following commands:

  ```
  1> use database_name
  2> go
  
  1> select data_pgs (8, doampg)
  2> from sysindexes where id = 8
  3> go
  ```

  This is the fastest way to determine how full the transaction log is. Note that this query may be off by as many as 16 pages.
- Use the **sp_spaceused** procedure on *syslogs*.

Thresholds and Transaction Log Management

SYBASE SQL Server release 10.0 introduced a new feature called the Threshold Manager. It consists of a set of functions that keep track of free space in a database and a means for users to manage that space. While thresholds may be used to manage space on any named segment in a database, this section will only discuss managing space on the log segment. A complete description of thresholds, including examples, is presented in the *System Administration Guide*.

A threshold has a **free space amount** and a **stored procedure** associated with it. When log segment free space falls below the free space amount, threshold will act as a trip wire, activating the associated stored procedure. The procedure may alert users, dump the transaction log, increase the log space in a database, or take some other action.

Last-Chance Threshold

Each database created with its log on a separate device in System 10 or later will have a **last-chance threshold** established. Databases upgraded from pre-System 10 may have last-chance thresholds established with the function **lct_admin**, which is discussed in “Functions Pertaining to Last-Chance Threshold” on page 2-14. The last-chance threshold established by SQL Server has a free-space amount that cannot be changed by a user. The free space is what the SQL Server needs to do a successful **dump transaction *database_name***. Please note that this is not **dump transaction *database_name* with**

truncate_only or **dump transaction *database_name* with no_log**, but a full dump of the transaction log which can be used for recovery if needed.

The amount of free-space for a last-chance threshold is calculated by SQL Server and varies depending on the size of the log segment. A larger log segment will have more free space associated with the last-chance threshold. If the log segment is modified in a database (**sp_logdevice** or **sp_extendsegment**), the last-chance threshold will be recalculated, and a new value will be established by the threshold manager.

Generally, the procedure associated with the last-chance threshold will be **dump transaction** in one form or another. If the last-chance threshold is reached and the procedure does not execute for some reason, all transactions trying to write to the transaction log will be “put to sleep” by SQL Server. They will show a status of “LOG SUSPEND” in **sp_who**. Once appropriate action is taken to free space in the log, the transactions will “wake up” and continue processing.

You can, alternatively, use the **abort tran on log full** database option to abort processes, rather than suspending them when the last-chance threshold has been crossed. Refer to the *System Administration Guide* for details.

The last-chance threshold and associated procedures can be used successfully to prevent 1105 errors (segment full) on the log segment. However, it is still possible to get such an error. If the procedure associated with the last-chance threshold is a full **dump transaction *database_name***, and there is not enough room in the transaction log to do this, the last-chance procedure will generate Error 1105, State 4. In this case, the only alternative is to execute **dump transaction** with the **truncate_only** or **no_log** options.

Functions Pertaining to Last-Chance Threshold

Two functions exist which relate to thresholds: **lct_admin** and **curunreservedpgs**. Both are discussed in detail in the *SQL Server Reference Manual*. They are discussed briefly below.

lct_admin deals exclusively with managing the last-chance threshold. Following is the syntax and description of each option associated with the function:

- **lct_admin ('reserve', *pages*)** – given a specific value, this returns the number of pages that define the last-chance threshold for a log segment of *pages*.

- **lct_admin** (**'lastchance'**, ***dbid***) – creates a last-chance threshold of predetermined size in a specified database. This option is used to establish last-chance thresholds for databases that were upgraded from pre-System 10 SQL Servers. It cannot be used to modify an existing last-chance threshold.
- **lct_admin** (**'logfull'**, ***dbid***) – this returns 1 or 0, depending on whether the last-chance threshold has been crossed in the specified database.
- **lct_admin** (**'unsuspend'**, ***dbid***) – "wakes up" all transactions that were put to sleep when the last-chance threshold was crossed. This option also prevents additional processes from being put to sleep. All transactions will continue to write to the transaction log.

curunreservedpgs is used to display the number of free-space pages of a given segment, including the log segment. This value is available in *sysusages.unreservedpgs* of the *master* database, but that value is frequently inaccurate. **curunreservedpgs** provides a more accurate value. The syntax is as follows:

```
curunreservedpgs(dbid sysusages.lstart, 0)
```

This displays the number of free pages on the segment for *dbid* starting on *sysusages.lstart*.

Threshold-Specific Stored Procedures

In addition to the last-chance threshold, users may add other thresholds to the log segment. Details about stored procedures are in the *SQL Server Reference Manual*.

Following is a brief description of threshold-specific stored procedures with information regarding last-chance thresholds:

- **sp_addthreshold** – used to add thresholds to any segment, including the log segment. This would be in addition to the last-chance threshold.
- **sp_dropthreshold** – used to drop any segment from any device, with the exception of the last-chance threshold on the log segment. Dropping the last-chance threshold is not allowed.
- **sp_helpthreshold** – this displays information on all thresholds, including the last-chance threshold, or about one specific threshold.
- **sp_modifythreshold** – used to modify parameters of a threshold. In the case of the last-chance threshold, only the procedure name

may be modified. No updates are allowed to the threshold name or free-space amount.

Threshold-Related Stored Procedures

Following is a list of stored procedures that have been modified for thresholds, with emphasis on their relation to the last-chance threshold:

- **sp_dboption** – two options have been added. They are:
    - **abort xact when log is full** – when the last-chance threshold is reached in a database, this option causes all transactions that write to the transaction log to abort or sleep, depending on whether the option is set to true or false in a given database.
    - **disable free space accounting** – this option turns off space accounting, effectively ignoring thresholds on all segments except the log segment.
- **sp_dropsegment** – if a log segment is made smaller with this procedure, SQL Server automatically recalculates and establishes a new free-space amount for the last-chance threshold.
- **sp_extendsegment** – if a log segment is extended, SQL Server automatically recalculates and establishes a new free-space amount for the last-chance threshold.
- **sp_logdevice** – this procedure is used to put the log segment on a separate device for a database that previously had data and log on the same device. SQL Server will calculate and establish a threshold for the log device and enter a row in *systhresholds*.
- **sp_who** – a status of “LOG SUSPEND” will be given to tasks that were trying to write to the transaction log when the last-chance threshold of a database was crossed.
- **sp_helpdb** – the free space for each segment will be displayed, including the log segment.

How to Dump Transaction Logs Automatically

Depending on the particular requirements of the database, the decision on whether or not to dump the log is based on such factors as:

- Percentage full
- Amount of space available
- Time since the last dump

The following pseudo code shows one way to dump a log that is 25 percent or more full:

```
every N minutes
    begin
        determine amount of syslogs used
        extract percentage of log used
        if log is more than 25% full
            dump the transaction log to a device
    end
```

The frequency with which this task is performed and the threshold value depend on the particular needs of the database. These values should be set so that the log is checked often enough that the transaction log does not fill up. However, be careful not to run the **dbcc checktable** command too often because it requires considerable overhead.

An automatic dumping process such as this, which uses a threshold value to determine when to dump a database, helps prevent unnecessary log dumps. For example, if the transaction log is dumped periodically, regardless of how full it is, some of the dumps may contain very little information and could possibly have been avoided.

3 Backup Server

This chapter describes how Backup Server works, its hardware and software requirements, and some common Backup Server problems, including a list and explanation of error messages.

What Is Backup Server?

Backup Server is an Open Server-based utility that handles all dumps and loads for SQL Server releases 10.0 and later. Backup Server provides flexible syntax for the **dump** and **load** commands and eliminates the need for the console program. Because it is a separate collection of processes, Backup Server does its tasks independently from the SQL Server, meaning that users experience much less performance degradation during dumps and loads.

How Does It Work?

Backup Server runs on the same machine as SQL Server (or on the same cluster, for Digital OpenVMS). You can perform dumps and loads over the network by using two Backup Servers, one on the local machine and the other on a remote machine.

When you **dump database/transaction** to dump devices known by a remote Backup Server (through **sp_addumpdevice**), the local Backup Server reads the database devices and sends the data over the network to the remote Backup Server, which stores it on the dump devices. When you use the **load** command, the local Backup Server sends instructions over the network to the remote Backup Server. The remote Backup Server reads the data from the dump devices and sends it back to the local Backup Server, which writes the data to the database devices. A network dump performs only as well as does the network supporting it, leading in many cases to significant performance decreases.

Backup Server contains several major new features to allow users to implement an unattended backup policy to improve dump and load performance. SQL Server is able to detect remaining space on database segments and allows automatic dumping and truncation of the transaction log when free space falls below particular thresholds. Backup Server:

- Senses dump device type and density automatically
- Supports writing multiple dumps to the same volume
- Supports dump striping (interleaving of dump data across several volumes)

Dump Striping

Backup Server supports simultaneous dumps and loads with a maximum of 32 backup devices, in parallel, **per dump or load**. The number of simultaneous dumps and loads is limited by the system's configured maximum number of open files, and by shared memory resources. On Digital OpenVMS, that is the per-process limit. On UNIX, there is a limit on the number of processes per user ID, and a system-wide limit, since UNIX forks a separate process for each device to do the I/O. Refer to the *System Administration Guide Supplement* for your platform for more information on the maximum number of open files.

Refer to *SYBASE SQL Server Utility Programs for UNIX* for more information on the Backup Server's configurable parameters:

- **-C** specifies the number of Server connections
- **-N** specifies the number of network connections

See the *SQL Server Reference Manual* for more information about **dump** and **load** syntax, particularly in the area of multi-file dumps to a single volume set and single database dumps across a multi-volume set.

See "Developing a Backup and Recovery Plan" in the *System Administration Guide* for more information on the capabilities of the Backup Server.

Compatibility Issues

An important goal of the System 10 **dump**/**load** architecture is to preserve compatibility with pre-System 10 dump scripts. The enhancements of the System 10 architecture do not invalidate existing dump scripts because the existing syntax is upwardly compatible with the new syntax. You will need to modify your dump handling techniques around the new scheme for handling volumes, but the modifications should be toward more convenience. For example, multiple dumps per tape volume are the default.

Compatibility is also an issue for transporting dump volumes from one computing platform to another. The objective of the Backup Server is to create maximally portable dumps with native algorithms. While Sybase does not officially support dumping on one platform and loading to another, the parameters known to require equivalence between platforms in order to transport dumps are:

- Physical media
- Machine byte ordering (CPU set)
- Database page size
- Floating-point format (if data contains floating point values)
- Database layout release (for example, 4.2 compared to 10.0)

◆ ***WARNING!***

The dump and load commands in release 10.0 and later releases are not compatible with dump volumes created with pre-10.0 SQL Servers. In addition, use of 10.0 or later dumps with pre-10.0 releases of SQL Server is not supported.

Shutdown

Backup Server supports immediate or deferred shutdown. You must connect to your SQL Server to issue the **shutdown** command to Backup Server.

Deferred Shutdown

The syntax for deferred shutdown is as follows:

```
1> shutdown backup_server_name
2> go
```

This shutdown permits dumps and loads in progress to complete (including volume change notifications for these sessions), but will not permit initiation of new dumps and loads. When the last dump or load has completed, Backup Server disconnects and exits from all SQL Servers.

Immediate Shutdown

The syntax for immediate shutdown is as follows:

```
1> shutdown backup_server_name with nowait
2> go
```

This causes Backup Server to drop all connections immediately and to exit.

➤ ***Note***

The **shutdown** command must be typed while you are logged into SQL Server. In the syntax examples above, *backup_server_name* is the local/remote Backup Server name known internally to SQL Server (that is, it is not the network name). If you are using only one Backup Server, the *backup_server_name* should be SYB_BACKUP.

What Is Required to Run Backup Server?

Backup Server is an Open Server application and an independent process. As such, it uses memory and CPU resources, and acts like an independent SQL Server.

Name Entries

sybinit inserts entries for Backup Server into the interfaces file and the *sysservers* table in the *master* database (where it is listed as "SYB_BACKUP"). If you have a remote Backup Server on another machine, put your interfaces file on a file server accessible to both machines, or else copy the remote Backup Server's entry from the remote interfaces file to the local one.

The definition for the remote Backup Server must exist in the interfaces file on both the local and the remote system.

◆ ***WARNING!***

The local Backup Server entry in sysservers must have an accurate network name. Run sp_helpserver to verify this. If entries are inaccurate, SQL Server may contact the wrong Backup Server and write to or read from the wrong devices.

Additionally, if the Backup Server start-up option **-S** specifies a particular server name, all interfaces files must use that name, not another. In other words, you cannot use an alias for your Backup Server in the interfaces file as you can for other servers. For example:

- If you have started a Backup Server with the **-SSYB_BACKUP** option, and
- You have *BS2* aliased to SYB_BACKUP in your interfaces file, and
- You attempt to do a dump to *dumpdev* at BS2,

then you will receive the following errors as SQL Server assumes you are dumping to a remote Backup Server:

```
Backup Server: 5.16.2.1: DB-Library error, error number 20011,
severity 8:
Maximum number of DBPROCESSes already allocated.
Backup Server: 5.3.2.1: Cannot open a connection to the slave
site 'REM_BACKUP_SERVER'

Backup Server: 5.16.2.1: DB-Library error, error number 20018,
severity 5:
General SQL Server error: Check messages from the SQL Server.
Backup Server: 5.7.2.4: RPC ('as_arch_device') execution failed.
```

UNIX Memory Requirements

Under UNIX, Backup Server uses shared memory during dump and load (Digital OpenVMS does not use shared memory, so these calculations do not apply). The size and number of shared memory segments, and the number of process attachments to those segments, vary depending on the number of stripes and whether the dump is remote:

- There is one session handler for each **dump** or **load** currently in progress.
- There is one stripe service task for each stripe being used by the **dump** or **load**.

Local Stripe

- The Backup Server process creates and attaches to one shared memory segment. The size of this memory segment is 2,048 bytes * the number of the local stripes.
- Each local stripe also creates a shared memory segment. The Backup Server process does not attach to this segment. Each local

stripe may allocate up to 110K of shared memory (164K on HP-UX). **sybmultbuf** is a process forked by the Backup Server which reads and writes the data from the database devices. Only the two **sybmultbuf** processes associated with each local stripe attach to this segment.

Remote Backup Server

The remote Backup Server requires different calculations for shared memory:

- The remote Backup Server allocates up to 54K of shared memory per stripe.
- The Backup Server process attaches to one shared memory segment per remote stripe. For example, if a local Backup Server dumps to a remote Backup Server with two stripes, the remote Backup Server creates two shared memory segments and attaches to both.

Shared Memory Calculations

Some platforms require a configuration parameter (*SHMSEG*) for the number of shared memory segments to which a process can attach. During a local dump or load, the **sybmultbuf** process attaches to one or two shared memory buffers, depending on whether it is reading (one buffer) or writing (two buffers) the data. The main Backup Server process only attaches to one buffer per active dump or load:

- Local dump
  - Backup Server process:
    2048 bytes * number of local stripes + (Backup Server process)
  - **sybmultbuf**, UNIX except HP:
    108K * number of local stripes
  - **sybmultbuf**, HP-UX:
    164K * number of local stripes

  Backup Server process attaches to one shared memory segment per active dump. The **sybmultbuf** process attaches to two.
- Remote site
  - UNIX except HP:
    54K * number of remote stripes (active at one time)
  - HP-UX:
    size = 54K * number of remote stripes (active at one time) + 64K

The Backup Server process attaches to one shared memory segment per remote stripe. The **sybmultbuf** process attaches to two.

System Defaults

The default number of shared memory segments a process can attach to varies from platform to platform. Refer to the *System Administration Guide Supplement* for your platform for this figure. If more than six stripes are to be dumped, the *SHMSEG* operating system configuration parameter needs to be increased.

Because Digital OpenVMS has true asynchronous I/O built in, the Backup Server itself does all I/O and there is no **sybmultbuf** equivalent. There is no shared memory because there are no subprocesses ("child" processes) communicating with each other.

Multi-File, Multi-Volume Dumps

The Backup Server allows writing of data from successive **dump**s to the same archive volume by default if the dump device type supports it. This feature makes efficient use of high-density archive media such as 8-mm tape.

The Backup Server senses the media type of the archive and determines whether the device:

- Supports random or sequential access
- Accommodates more than one dump file per volume
- Allows dismounting of a filled volume and mounting of a new volume

Operating system files and raw disk partitions are **single-volume**, **single-file** media. This means that these devices can contain, at most, one Sybase dump and that the dump resides entirely within the file or partition.

QIC tapes and removable disks are all **single-file**, **multi-volume** media. This means that a dump may span multiple volumes, but a set of volumes may collectively contain, at most, one dump. These devices do not allow backspacing, and attempts to append multiple files will prompt you for overwrite confirmation and will overwrite the previous contents if you enter "PROCEED."

DAT, 8-mm, and 9-track tapes are **multi-file**, **multi-volume** media. This means that a given volume of one of these media types may contain more than one dump, and the last dump on a volume may continue onto other volumes.

If you specify the **INIT** option for a stripe, that stripe reinitializes and overwrites all volumes it uses, without regard to their contents and without prompting you for confirmation.

Tape Device Hardware Requirements

The following table lists device names and specific hardware requirements for **dump/load** to and from a tape device. The *System Administration Guide Supplement* for your platform may have further information. In the table below, ***N*** stands for the device number.

Table 3-1: Device names for database dumps to tape

Platform	Type	Device Name
HP9000	4-mm (/ SCSI and / HPIB)	*/dev/rmt/**N**mn*
	9-track (/ SCSI and / HPIB)	*/dev/rmt/**N**mn*
AT&T	8-mm / SCSI (5GB and 2.2GB)	*/dev/nrmt**N***
	4-mm / SCSI	*/dev/nrmt**N***
RS6000	9-track / SCSI	*/dev/rmt**N**.[0-127]*
	8-mm DAT / SCSI (5GB and 2.2GB)	*/dev/rmt**N**.[0-127]*
	QIC / SCSI (1 / 4-inch cartridge)	*/dev/rmt**N**.[0-127]*
SunOS 4.x (BSD)	9-track / SCSI	*/dev/nrmt**N***
	QIC / SCSI (1 / 4-inch cartridge)	*/dev/nrar**N***
	8-mm / SCSI (5GB and 2.2GB)	*/dev/nrst**N***
Sun Solaris 2.x (SPARC)	9-track	See “Sun Solaris 2.x” on page 3-10.
	1 / 4-inch cartridge	
	8-mm / SCSI (5GB and 2.2GB)	
Digital OpenVMS VAX	9-track (/ HSC and / DSSI)	See “Digital OpenVMS” on page 3-10.
	8-mm (/ HSC 5GB and 2GB and / DSSI)	
	TK50 (/ HSC and / DSSI)	

Table 3-1: Device names for database dumps to tape (continued)

Platform	Type	Device Name
Digital OpenVMS Alpha	8-mm / SCSI	See “Digital OpenVMS” on page 3-10.
	9-track	
	4-mm DAT / SCSI	
Digital UNIX	4-mm DAT / SCSI	See “Digital UNIX” on page 3-10.
	TZ85I	
Novell Netware	1/4–inch cartridge	***N*** (from the **list device** command)
	8-mm	***N*** (from the **list device** command)
OS/2		*e:\tmp\dbase.dmp*
Windows NT	1/4-inch cartridge	\\.*TAPE**N***
	4mm DAT	\\.*TAPE**N***
	8mm DAT	\\.*TAPE**N***
SCO UNIX	1/4-inch cartridge QIC	*/dev/ct0* or */dev/Stp0*
	1/4-inch SCSI QIC	*/dev/Stp0*
	4mm SCSI	*/dev/Stpdevice#*
	8mm SCSI	*/dev/Stpdevice#*

➤ ***Note***

QIC devices are single-file devices. They do not allow backspacing and attempts to append multiple files to such a device will prompt you for overwrite confirmation.

System 10 requires non-rewinding devices so that Backup Server can control the positioning of the tape device.

Sun Solaris 2.x

For Sun Solaris 2.x environments, tape device names are constructed as follows:

/dev/rmt/ unit_number density [BSD behavior] no rewind

where *density* is “l” (low), “m” (medium), “h” (high), or “c” (compressed); and *unit_number* is a logical number that uniquely identifies the tape drive or unit.

Tape device names indicating BSD behavior are used during installation, but should **not** be used with Backup Server.

Digital UNIX

For Digital UNIX environments, tape device names are constructed as follows:

/dev/[n]nrmt/unit_number density

where *density* is “a” or “l” (low), “m” (medium), or “h” (high); *[n]* indicates “no rewind”; and *unit_number* is a logical number that uniquely identifies the tape drive or unit.

Digital OpenVMS

For Digital OpenVMS environments, there is no standard device name for tape devices. Some typical names are: *mua0*, *mub1*, *msa0*. The devices are named as follows:

type spec unit

where:

- *type* is the controller type
- *spec* is the controller specifier in the backplane
- *unit* is the unit number on the controller

For example, tape device *mub1* is unit *1* of the *b* controller of an *mu* type controller.

HP-UX

For HP-UX environments, tape device names are constructed as follows:

/dev/rmt/controller_number device number density [compression]no rewind

where *density* is “l” (low), “m” (medium), or “h” (high).

Localization Files

Backup Server is an Open Server™ application, so it requires the *$SYBASE/locales/locales.dat* (*[SYBASE.LOCALES]LOCALES.DAT* on Digital OpenVMS) files that are required by Open Server, plus some additional files for each character set. Here is the list for the us_english, iso_1 character set:

```
locales/us_english/iso_1/bslib.loc
locales/us_english/iso_1/common.loc
locales/us_english/iso_1/cslib.loc
locales/us_english/iso_1/dblib.loc
locales/us_english/iso_1/libdna.loc
locales/us_english/iso_1/libinsck.loc
locales/us_english/iso_1/libsdna.loc
locales/us_english/iso_1/libssri.loc
locales/us_english/iso_1/libtli.loc
locales/us_english/iso_1/tcllib.loc
locales/us_english/iso_1/oslib.loc
```

➤ ***Note***

libsdna.loc is required for Digital OpenVMS only; *libtli.loc* is required for Sun Solaris 2.x only.

Identical sets of files are located in each of the character set and language directories:

```
locales/us_english/cp437/
locales/us_english/cp850/
locales/us_english/iso_1/
locales/us_english/mac/
locales/us_english/roman8/
```

Only U.S. English is provided with the basic product. If you have purchased one of the language modules, the same set of locales files is provided for each character set supported for the language.

Following are the locations of files needed for some other languages:

Japanese

```
locales/japanese/deckanji/
locales/japanese/eucjis/
locales/japanese/sjis/
locales/us_english/deckanji/
locales/us_english/eucjis/
locales/us_english/sjis/
```

French

```
    locales/french/cp437/
    locales/french/cp850/
    locales/french/iso_1/
    locales/french/mac/
    locales/french/roman8/
```

German

```
    locales/german/cp437/
    locales/german/cp850/
    locales/german/iso_1/
    locales/german/mac/
    locales/german/roman8/
```

Individual locales file names are the same for Digital OpenVMS, with paths like the following:

```
  [SYBASE.LOCALES.US_ENGLISH.ISO_1]BSLIB.DOC
```

Additionally, Backup Server requires the file *charset.loc* in at least the default character set subdirectory (*charsets/iso_1/charset.loc*) and at least the default sort-order file (*charsets/iso_1/roman8.srt* for HP; *charsets/iso_1/binary.srt* for other platforms).

By default, Backup Server uses the *$SYBASE/locales/locales.dat* file's entry for *platform default* as the locale. You can override this with the LANG shell variable or any of the Open Server language variables. With SQL Server release 10.0.1, you can use the **backupserver** command's **-J** option (the option is **/character_set** for Digital OpenVMS) to define the Backup Server's character set. See the **backupserver** entry in the *SQL Server Utility Programs* manual for your platform for more details.

➤ ***Note***

Backup Server may issue a message informing you that SQL Server's character set is different from Backup Server's, which may lead you to worry about database corruption if you are doing local or remote backups to Backup Servers with different character sets. However, there is no need for concern. This is only an informational message.
Backup Server does not convert the data pages from the Server database device when performing dumps or loads. Localization changes are made only for messages passed back to the client. The rules are:

1. Backup Server sends messages to SQL Server in SQL Server's character set but in the client's language.

2. Backup Server reads pages from the database and writes them to an archive device, and vice versa. It does not interpret the data. It only copies it. No pages are ever changed while they are being dumped or loaded.

Additional Requirements

- SQL Server must be configured for remote access. You may need to restart SQL Server if you have changed from this default behavior.
- On UNIX systems, the user (usually "sybase") who starts the Backup Server process must have write permission for the */tmp* directory.
- The user who starts the Backup Server process must have write permission for the dump devices and read permission for the database devices.

Common Problems

Following are descriptions of some common problems that occur when you use the Backup Server.

"No language handler installed" Message

If you connect directly to the Backup Server with **isql** and attempt to issue a command, you will get messages like the following:

```
1> dump database master to tapedump1
2> go
```

```
No language handler installed.
Language cmd: dump database master to tapedump1
```

This is expected behavior because Backup Server is not a standalone SQL Server. It is an Open Server application, is not intended to accept direct commands, and, therefore, has no way of parsing them. All commands to the Backup Server, such as **shutdown** or **dump** and **load**, can be issued only from a running SQL Server.

LOGCONN Errors in SQL Server After Using Backup Server

The SQL Server site handler reports this error when the server has unexpectedly lost contact with another site. If the other site is a Backup Server, then a dump or load session has prematurely completed. This error message appears in the SQL Server error log:

```
site_hdlr: No LOGCONN for packet from site 1,
channel 2
```

When the error is reported because a dump or load session was aborted by a user interrupt (such as Ctrl-c), the message should be treated as an informational message.

This error can also be reported if the SQL Server unexpectedly loses contact with the Backup Server. In this case, the error should be investigated.

load {database | transaction} with listonly Locks Database

A database being loaded into is locked so that no other users can use it, even though no actual data may be loaded, as in the case of a **load with listonly.** To work around **load with listonly** locking an active database, create a dummy database and do the **load with listonly** into that database instead of the database you do not want to have locked.

Some Messages from Backup Server Are Lost

Messages initiated by a threshold procedure may be lost if the boot window is not present. SQL Server has no mechanism for intercepting a Tabular Data Stream (TDS) received from other servers; the TDS is merely passed on to the client. Threshold procedures have no “client,” so messages from Backup Server to these tasks will not appear unless you specify otherwise. You must specify **notify=operator_console** if the dump is invoked by a stored procedure, including **sp_thresholdaction**; this is an issue for volume change prompts, for example, to which you cannot respond if you never see them.

dump database on Digital OpenVMS Alpha/Digital UNIX Raises Error

Attempting to use **dump database** when your database devices are on raw partitions (DEC HSC [mscp] devices) raises the following error:

```
Backup Server: 4.80.2.1: Server, device
/dev/44a0e: You must use the no-rewind-on-close
tape device.
```

This is not a raw device problem in general, but a DEC HSC (mscp) device issue. Digital UNIX has a known problem on its HSC device driver that causes Backup Server to fail to identify the HSC device type. Digitals's CLD #00266 states that such devices return 0 when queried with "MTIOCGET ioctl."

◆ ***WARNING!***

Sybase currently supports only SCSI devices on Digital UNIX.

kill Does Not Terminate Some Processes

If a **dump** process does not terminate upon receipt of the **kill** command, check its state and your Backup Server release. Processes in a state of "LOG SUSPEND" do not terminate in releases prior to 10.0.1.

Open Server Error Messages Related to Backup Server

Following are some Open Server error messages that can appear when you are using Backup Server.

Error 5704.10.0

```
Open Server Error: 5704.10.0: Changed client
character set setting to 'cp850'
Open Server Session Fatal Error: 16227.15.0:
Unknown token TDS stream received by spid 5
```

This message can appear if a user, who succeeds at a local **dump database**, tries to dump to a remote Backup Server.

Check your default character set. If the default character sets for the remote Backup Server and SQL Server are different, change the Backup Server's default character set, restart the Backup Server, and rerun the **dump**.

Error 16227.15.0

```
Open Server Session Fatal Error: 16227.15.0:
Unknown token TDS stream received by spid <spid #>
```

This error occurs when you are trying to use a 10.0 Backup Server with a 10.0.1 SQL Server. Check the release string of your Backup Server; if it is 10.0, switch to a 10.0.1 Backup Server and retry the load. 10.0.1 SQL Server and the compatible 10.0.1 Backup Server should be able to load dumps from 10.0.

Error 16240.20.0

Many error conditions are associated with this Open Server error, differentiated by the accompanying messages. The following errors begin with the phrase "Open Server Fatal Error 16240.20.0":

```
Net-Library routine net_address_get() failed in
srv__start_net
Network error: status = 12 - Specified server name
attribute could not be found
Backup Server: 1.29.2.1: Unable to start the
Backup Server. See Backup Server error log for
additional information.
```

This is the Open Server equivalent of "Server name not found in interfaces file." The Backup Server will not start. Check to be sure the name of the server you are trying to start is in the interfaces file, and remember that server names are **case-sensitive**.

SYB_BACKUP is the **logical** name of the Backup Server. It may also be the **physical** name, but it need not be.

Under normal circumstances, **sybinit** sets up the Backup Server, but if your Backup Server name has been edited or deleted from the interfaces file, add it again with **sp_addserver**, as follows:

```
1> sp_addserver SYB_BACKUP, NULL, physical-name
2> go
```

The specified *physical-name* is the name to use, both in the interfaces file and when you do a **dump** or **load** with the **at** option.

Errors in Backup Server Error Log

The following error messages may appear in the Backup Server error log.

EMULATOR DEAD

```
EMULATOR DEAD
```

This error can occur on HP-UX and DG / UX machines running 10.0 Backup Server; the problem is fixed in the 10.0.1 release. If you are running a 10.0 Backup Server, try decreasing the *max file descriptors* kernel parameter. This usually solves the problem.

On DG / UX there are two parameters for file descriptors:

- *SDESLIM* – soft limit
- *HDESLIM* – hard limit

To work around the "dead emulator" problem, modify the value of *HDESLIM*:

- with *HDESLIM* = 2048 it does not work
- with *HDESLIM* = 1024 it does work

Failed to open database into file

```
Failed to open database into file
/tmp/SYB_BACKUP5: No such file or directory
EMULATOR ERROR: Cannot start the emulator. No such
file or directory
```

This message appears in *srv.log*. On the console, you will see:

```
backup server internal error: 4.62.3.5:
Multibuffering subprocesses died, archive
/usr/sybase/dump/master/master.dump
```

There are a number of reasons for emulator failure. Most of the reasons have accompanying error messages that correctly state the problem. However, in this case, the message is misleading and actually stems from a UNIX Backup Server having problems when the machine it is running on is configured for more than 1280 file descriptors. If possible, reduce the number of per-process file descriptors. Also, check your release number for your Backup Server. This problem has been fixed in the 10.0.1 release.

Net-Library routine *net_dict_open* failed in *srv__open_dictionary*

```
Net-library routine net_dict_open() failed in
srv__open_dictionary()
Network error: status = 11 - Could not find
addressing dictionary. No server log file open;
Using stderr for log.
```

The interfaces file does not exist or cannot be opened. Check to be sure the file exists and the permissions are set correctly.

Net-Library routine *net_listen* failed in *srv__start_net*

```
Net-Library routine net_listen() failed in
srv__start_net.
Network error: status = 12 - Net-lib protocol
driver call to register a listener failed
```

This is the Open Server equivalent of "Network port already in use."

If your system is Digital OpenVMS using DECnet:

- The SYSNAM privilege is required to start a DECnet listener, just as with SQL Server. Check to be sure this privilege is set appropriately.
- If this fails to help, try using DECnet object names instead of object numbers. If this works, then report to Sybase Technical Support that you have encountered bug 51237.

Net-Library routine *net_init* failed in *srv__init_net*

```
Net-Library routine net_init() failed in
srv__init_net()
Network error: status = 162 - Unable to determine
Net-Library error.
No server log file open; Using stderr for log.
```

Localization information is not as expected, possibly the result of loading an incompatible Sybase product release into the same *$SYBASE* directory as the Backup Server. Be sure the entire release directory is up to date and that you have not accidentally renamed or deleted localization files.

No driver of the requested protocol class is available

```
No driver of the requested protocol class is
available.
```

This message indicates that the entry in the interfaces file for the requested Backup Server is wrong. Refer to the *System Administration Guide Supplement* for your platform to find out the correct interfaces file entries.

Net-lib protocol driver call to connect two endpoints failed

```
Net-lib protocol driver call to connect two
endpoints failed.
```

An error occurred while trying to connect to the specified Backup Server. Make sure the device name in the interfaces file is correct and that the Server's network address is correct in the interfaces file.

Error Conditions Without Messages

A programming error may occur that causes the Backup Server to hang without issuing a message. Follow these steps:

1. Determine whether it is a single session or the entire Server that has hung.
2. Terminate the hung session with keyboard interrupt or **quit**.

At this point, on a UNIX machine, it is necessary to check for I/O subprocesses that remain alive, holding open dump devices. Perform this check with the **ps** command and use **kill** (default signal) to kill the **sybmultbuf** processes whose command lines contain device names used by the terminated session. Be careful to kill only those processes that use the devices involved.

Each device is controlled by a pair of processes, and each pair of processes controls just one device. Therefore, the presence of a device name in a command line is sufficient to identify the subprocess to kill. **dump** and **load** data from an aborted session or Server are not usable.

The Backup Server can exit abnormally without killing its subprocesses. In this case, use **kill** to clean up the **sybmultbuf** processes before restarting the Backup Server; otherwise, the devices involved may not be usable.

Backup Server Error Messages

Error Number Format

A Backup Server error number has four components:

major.minor.severity.state

major numbers map to functional areas as follows:

Table 3-2: Backup Server major error numbers

Major Number	Description
1	Critical errors
2	Open Server event errors
3	Backup Server RPC API errors
4	I/O service layer errors (that is, operating system I/O calls)
5	Network data transfer errors
6	Archive volume manipulation errors
7	Parser errors (errors in the **dump** or **load** command options)

minor numbers order the errors within a major category. There is no special meaning to these numbers; they serve only to distinguish between errors.

severity codes are as follows:

Table 3-3: Backup Server severity codes

Severity Code	Description
1	Information, no user action necessary.
2,3	An unexpected condition, possibly fatal to the session, has occurred. Error may have occurred with any or all of usage, environment, or internal logic. This severity consists of two levels, which indicate whether or not the session must exit. 2 indicates the session must exit. 3 indicates the session may or may not exit.
4	An unexpected condition, fatal to the execution of the Backup Server, has occurred. The session must exit immediately.

state codes have a one-to-one mapping to instances of the error reported within the code. They are omitted in *Table 3-4: Backup Server error messages*, for clarity.

The following table lists Backup Server error messages.

Backup Server Error Message Descriptions

Table 3-4: Backup Server error messages

Error #	Message	Explanation
1.1.2	Memory allocation failed in %1! for a %2! allocation. This DUMP or LOAD session must exit.	Indicates shortage of virtual memory on the machine running Backup Server. User should terminate unnecessary processes.
1.5.4	Failed to install CS-Library message callback routine.	Backup Server's attempt to install an event handler failed. Backup Server cannot run.
1.7.1	Cannot allocate the locale info of the task. Use the Backup Server default language '%1!' instead.	The local structure for the client connection to Backup Server could not be allocated, and the thread property for the local information could not be gotten. Refer to the Backup Server error log for more information.
1.14.2	Unrecoverable I/O or volume error. This DUMP or LOAD session must exit.	This message is usually preceded by a more specific message describing a fault encountered in the lower I/O layers. One exception: this message is the only error indication when the Backup Server fails to start the multibuffering subprocess that it uses to perform backup I/O. Reasons for this failure are the non-existence of the **sybmultbuf** program in the standard locations (**-M** option argument, Backup Server's current directory or *$SYBASE/bin*). User should check the installation. If the **sybmultbuf** program can be installed in the current directory or *$SYBASE/bin*, it is not necessary to restart the Backup Server.
1.15.4	UNRECOVERABLE CONDITION: ALL SESSIONS WILL TERMINATE ABNORMALLY. THE BACKUP SERVER MUST EXIT.	This message is preceded by other messages describing the failure.
1.16.2	Error (major = %1!, minor = %2!) raised with unknown severity %3!.n	Indicates a programming bug in an instance of error reporting.
1.17.2	Major = %1!, Minor = %2!, Severity = %3!, state = %4!: error text is missing.	Indicates that localization files are out-of-date with the code. Check installation with Sybase Technical Support.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
1.18.1	WARNING: Requested %1! file descriptors, was allocated %2!; Backup Server will run out of file descriptors if this number is less than the number of database devices plus twice the number of stripes in concurrent use.	The Backup Server attempts to obtain the highest number of file descriptors allowed to a user program. This message indicates that the Backup Server found a lower quota. Concurrent activity must be restricted as directed or the platform administrator can increase the *HDESLIM*.
1.19.4	Cannot allocate the configuration structure.	Indicates either a basic Open Server failure or an extreme shortage of memory and swap space.
1.20.4	CS_CONTEXT allocation failed.	The context allocation routine failed when it tried to load localization files. The SYBASE environment variable is set incorrectly or the localization files are missing.
1.21.2	Unable to configure the maximum number of user events to %1!.	An error occurred when you tried to configure the Open Server's run-time maximum number of application-specific defined event types. Refer to the *Open Server Server-Library Reference Manual* for an explanation of events. Refer to the Backup Server error log for more information, and contact Sybase Technical Support.
1.22.2	Unable to configure the maximum number of SRV_PROCS to %1!.	An attempt to configure the Open Server run-time values to allow the specified number of Server threads failed. Backup Server uses one thread per dump or load connection plus an additional thread for each stripe. The configuration value is hard coded to 48.
1.23.2	Unable to configure the maximum number of network read buffers to %1!.	An attempt to configure the Open Server run-time values to allow the specified number of network read buffers failed. These buffers are used for Server to Server connections, and the configuration value is hard coded to 10.
1.24.2	Unable to configure the maximum number of connections to %1!.	Attempt to configure the Open Server run-time values to allow the specified number of network connections failed. The value of the **-C** option (**/connections** in Digital OpenVMS) to the **backupserver** command is used for the configurable value. Check the line in your *RUN_servername* file and see the entry for **backupserver** in *SQL Server Utility Programs* for your platform.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
1.25.1	Logging Backup Server messages in file '%1!'	Indicates that Backup Server messages are being written to the file named.
1.26.1	Unable to open Backup Server message log file '%1!'	Check the path to the message log file. Make sure that the SYBASE environment variable is set correctly.
1.27.1	Operating system swap space is low. The operating system may kill the Backup Server.	This is an RS6000 error only. The operating system will kill "random" processes in an effort to free space for critical processes. This message indicates that the Backup Server could be chosen. If necessary, reconfigure the Backup Server process as critical.
1.28.2	Open Server routine %1! failed. See Backup Server error log for additional information.	Internal error, contact Sybase Technical Support. This is a catch-all message for any Open Server function that raises an error for which Backup Server has determined the affected routine but not the exact error.
1.29.2	Unable to start the Backup Server. See Backup Server error log for additional information.	At Backup Server initialization time, the attempt to put Backup Server into a state in which it can run failed. Refer to any Open Server messages that are displayed.
1.30.2	Backup Server encountered error freeing memory.	Internal error, contact Sybase Technical Support. An error occurred when an attempt was made to free a block of memory from Backup Server's run-time heap. Either the memory was previously freed or there is another error in the Open Server routines for freeing memory. Check the Backup Server error log for any Open Server messages pertinent to this problem.
1.31.2	Unable to configure the maximum number of network connections to %1!. The number of network dump/load stripes that the Backup Server can run simultaneously is %2!.	Attempt to set the maximum number of network connections (DBPROCESSes) that the Server can originate failed. Refer to the usage of the **-N** command line option. It may be too large for DB-Library to handle.
1.32.2	Unexpected failure in 'cs_strcmp()' routine.	Internal error, contact Sybase Technical Support. An error occurred when comparing two strings to see if they are equal, greater than, or less than each other. The cause of the error could be an illegal string in length or content or that the character sets of the two strings are incompatible for comparison.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
1.34.1	Unable to set one or more trace flags; continuing.	Attempt to set the Open Server run-time trace flags failed. Check to see that the trace flags are valid and in the proper format.
1.36.1	Unable to configure Asynchronous I/O. If SQL Server is using Asynchronous I/O on block special devices, then dump and loads may be inconsistent. Refer to the server release notes.	HP only: could not initialize and configure the asynchronous I/O system to be used when accessing database devices. Check the Backup Server error log and contact Sybase Technical Support.
1.37.4	Could not allocate event flag, server %1!: code %2! message %3!.	This is an Digital OpenVMS error only. An attempt to allocate an event flag failed. Backup Server cannot start. Probably an internal error.
1.38.4	Could not get master lock on server name, server %1!: code %2! message %3!.	Digital OpenVMS only: an attempt to obtain a lock identifying this as the “master” Backup Server failed for the reason specified in %3!. Backup Server cannot start.
1.39.4	No server name given	Digital OpenVMS only: no Server name was given when attempting to acquire an exclusive lock on the master lock name, which is constructed from the Server name. Use **showserver** to check whether Backup Server was started with a Server name by means of the **/SERVER_NAME** qualifier or the translation of the DSLISTEN logical name.
1.41.4	A server named %1! is already running. Could not obtain stop-lock.	Digital OpenVMS only: another Backup Server with the same logical name is already running. This Backup Server cannot start.
1.42.4	STOP semaphore failed, code %1! message %2!.	Digital OpenVMS only: failed to queue the lock required to shut down Backup Server via **stopserver**. Backup Server cannot start.
1.44.1	Cannot set the server language to ‘%1!’	Attempt to set the Backup Server’s language failed. Refer to the Backup Server error log for more information. The specified language may not be a valid language, or the language may not be installed under *$SYBASE/locales*.
1.45.1	Cannot set the server character set to ‘%1!’	Attempt to set the Backup Server’s character set failed. Refer to the Backup Server error log for more information. The specified character set may not be a valid character set or the character set may not be installed under *$SYBASE*.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
1.46.1	CSLIB routine %1! failed. See Backup Server error log for additional information.	This is the catch-all message for Client-Server Library functions that return an error. Currently, the only reference to this error occurs when Backup Server is attempting to configure the localization properties of the process. Refer to the Backup Server error log for more information There could be something wrong with the specified language or character set, for example the specified character set is not valid for the specified language. Contact Sybase Technical Support if a resolution cannot be determined.
1.47.1	Cannot set language / charset to '%1! / %2!'	Attempt to set the Backup Server's language and character set failed. Refer to the Backup Server error log for more information. The specified language or character set may not be valid or the language or character set may not be installed under *$SYBASE/locales*.
2.1.2	Open Server initialization failed. Unable to start %1!.	The user should check the installation. This error can also arise from memory and swap space shortage.
2.2.4	Open Server Server Fatal Error: %1!.%2!.%3!: %4!	This indicates a fault within the Open Server layer that forces program exit. The error code placeholder should be matched to the Open Server error codes for diagnosis. Check installation.
2.3.2	Open Server Session Fatal Error: %1!.%2!.%3!: %4!	Interpret as for 2.2.4, except only the session exits, not the whole Server.
2.4.2	Open Server Recoverable Error: %1!.%2!.%3!: %4!	Interpret as for 2.2.4, except Server and session processing continue. Example: missing localization file.
2.5.2	Couldn't define %1! event; Backup Server must exit.	Indicates Open Server initialization failure or session limit reached.
2.6.2	Login information unavailable.	The Backup Server could not access the session login information to perform authentication due to unauthorized session initiator or programming bug. Contact Sybase Technical Support.
2.7.2	Remote server TDS version must be equal to or greater than 4.6. This session must exit.	Connection was attempted with an obsolete TDS version. Backup Server communicates reliably only with System 10 SQL Servers and other Backup Servers.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
2.8.2	Couldn't create semaphore for session %1!. This session must exit.	Indicates Open Server MUTEX initialization failure or session limit reached.
2.9.4	The number of connections must be an integer.	Indicates illegal **-C** (**/connections** in Digital OpenVMS) option value for the **backupserver** command. Check the line in your *RUN_servername* file.
2.10.2	A trace flag specifier must be an integer.	UNIX only: The argument to the **backupserver** command's **-T** (**/trace** in Digital OpenVMS) option was not an integer. Check the line in your *RUN_servername* file.
2.11.2	Failed to get lock to send packet to client, spid: %1!, lockstatus %2!.	Internal error, contact Sybase Technical Support. An error occurred when attempting to lock an Open Server MUTEX (mutually exclusive object) which only one user at a time can lock. Refer to the Backup Server error log for more information.
3.1.2	Unrecognized RPC received--ignored.	The session initiator sent an unrecognized RPC. Indicates ad hoc RPC was sent, or a programming bug in SQL Server or Backup Server. Contact Sybase Technical Support.
3.2.2	Symbol %1! not found in TDS table.	Internal message; customers will likely never see it; it is only included here for completeness.
3.3.2	%1!: May not specify Backup Server scanning or free-page clearing for a secondary phase.	Internal error, contact Sybase Technical Support. These errors indicate a violation of the dump synchronization protocol of the RPC API.
3.4.2	%1!: Must specify Backup Server scanning in order to clear free pages at LOAD time.	Same as Error # 3.3.2
3.5.2	%1!: Could not lock session %2! to begin phase.	Internal error, contact Sybase Technical Support. This indicates an error or bug in the Open Server MUTEX mechanism.
3.6.1	%1!: Phase %2! is currently active. Only one phase may be active at a time.	Internal error, contact Sybase Technical Support. Violation of the dump synchronization protocol of the RPC API.
3.7.2	%1!: phase %2! already concluded.	Same as Error # 3.6.1
3.8.2	%1!: May not begin phase %2! before completing an earlier phase, number %3!, for which there is uncompleted work.	Same as Error # 3.6.1

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
3.9.2	%1!: May not initiate Backup Server scanning for a phase for which run-lists have been queued.	Same as Error # 3.6.1
3.10.2	%1!: phase %2! not active.	Same as Error # 3.6.1
3.11.2	%1! SANITY: PHASE for phase no. %2! not at head of list.	Same as Error # 3.6.1
3.12.2	May not submit RPC %1! unless performing a DUMP.	Same as Error # 3.6.1
3.13.2	%1!: Illegal phase number %2!.	Same as Error # 3.6.1
3.14.2	Logical page %1! is not in the given database map.	Same as Error # 3.6.1
3.15.2	The disk piece for page %1! does not belong to phase %2!.	Same as Error # 3.6.1
3.16.2	%1!: There is no session of id %2! in progress.	Same as Error # 3.6.1
3.17.2	Pathname parameter must specify an absolute pathname. Rejected value: %1!	SQL Server sent its present working directory to the Backup Server and that directory was not recognized as an absolute path specifier.
3.18.2	SQL Server did not specify absolute pathname root before sending the relative device pathname %1!.	SQL Server sent a device specification that was not recognized as an absolute path specification, and had not previously sent its present working directory.
3.19.2	Combining device pathnames %1! and %2! exceeds the maximum pathname length.	The combination of SQL Server's present working directory and the relative device name is too long for Backup Server's internal buffer.
3.20.2	%1!: RPC rejected--missing parameters.	The number of parameters passed to Backup Server for the specified RPC does not match the number of parameters expected for that RPC. If you are sending Backup Server the RPC via your own customized mechanism, check that mechanism for the correct RPC count. Otherwise, contact Sybase Technical Support.
3.21.2	%1! SANITY: run list size > %2!.	The total size of the run lists for the *bs_run_list* RPC will exceed the size used to store the run lists internally within Backup Server. If you are sending Backup Server the RPC via your own customized mechanism, check that mechanism for the correct size of the run lists. Otherwise, contact Sybase Technical Support.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
3.22.2	%1!: Session %2! does not exist.	Internal error, contact Sybase Technical Support. Violation of the dump synchronization protocol of the RPC API.
3.23.2	%1!: Could not queue run list for session %2!.	Same as Error # 3.22.2
3.24.2	%1!: RPC rejected, phase %2! already concluded.	Same as Error # 3.22.2
3.25.2	%1!: May not submit run lists for phases in which Backup Server conducts the scan.	Same as Error # 3.22.2
3.26.2	%1! SANITY: parameter data overflows I/O block.	Internal error, contact Sybase Technical Support. Error with **dump** header or trailer processing in either Server. The total parameter sizes are defined to equal the I/O block size.
3.27.2	Invalid RPC sequence at RPC %1!.	Internal error, contact Sybase Technical Support. Error in the session initiator (SQL Server or Backup Server), a violation of the RPC API.
3.28.2	Non-contiguous database map, session %1!, lpn %2!.	Internal error, contact Sybase Technical Support. These errors indicate a SQL Server programming bug, defective database disk mapping information sent.
3.29.2	No virtual mapping for logical page %1!.	Same as Error # 3.28.2
3.30.2	Bs_normdbmap received an overlapping disk piece.	Same as Error # 3.28.2
3.31.2	Improper parameter count %1! for RPC %2!.	Internal error, contact Sybase Technical Support. Error in the session initiator (SQL Server or Backup Server), a violation of the RPC API.
3.32.2	Incorrect type for parameter #%1! for RPC %2!; expected %3! got %4!.	Same as Error # 3.31.2
3.33.2	SANITY: premature last run list.	Internal error, contact Sybase Technical Support.
3.34.2	SANITY: more stripes than extents.	Same as Error # 3.33.2
3.35.2	RPC %1! refused following previous FAILURE return.	Internal error, contact Sybase Technical Support. Error in the session initiator (SQL Server or Backup Server), a violation of the RPC API.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
3.36.2	RPC %1!: parameter %2! may not be NULL.	Same as Error # 3.36.2
3.37.2	SANITY: RPC %1!: expected parameter %2! got parameter %3!.	Same as Error # 3.36.2
3.38.2	There is no primary phase %1! for secondary phase %2!.	Internal error, contact Sybase Technical Support. Violation of the dump synchronization protocol of the RPC API.
3.39.2	RPC %1! can only be executed in a slave server.	These errors indicate a SQL Server programming error, violation of the RPC API.
3.40.2	Illegal length for parameter #%1! for RPC %2!; expected %3! got %4!.	Internal error, contact Sybase Technical Support. Error in the session initiator (SQL Server or Backup Server), a violation of the RPC API.
3.41.2	Illegal max-length for parameter #%1! for RPC %2!; expected %3! got %4!.	Same as Error # 3.40.2
3.42.1	%1! is complete (database %2!).	Informational message that Backup Server processing for the session is complete (the **dump** or **load** command may continue to run, for example, to clear the region of the loaded database above the dump-time high page).
3.43.1	Dump phase number %1! completed.	Informational message that a dump synchronization phase has completed.
3.44.2	The Backup Server is already performing a deferred shutdown; only one may be active at a time.	Shutdown is in progress and will occur when all dumps and loads have completed.
3.45.1	Attention received: canceling deferred shutdown.	Internal error, contact Sybase Technical Support.
3.46.2	Deferred shutdown wait failed, cancelling deferred shutdown.	Same as Error # 3.45.1
3.47.2	The Backup Server is undergoing shutdown. Your session will terminate immediately. Please reattempt connection later.	This message is issued in response to a **dump** or **load** command after a **shutdown** request has been sent but before the Backup Server has actually shut down.
3.48.1	The Backup Server will go down immediately. Terminating sessions.	The final message sent to the session that issued the **shutdown** command.
3.49.2	RPC %1! does not allow option %2!.	An RPC inconsistency was detected. This could be caused by incompatible releases of Backup Server and SQL Server.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
3.50.2	Non-privileged connection is not authorized to execute privileged RPC ‘%1!’.	Some RPCs are “privileged,” that is, they can only occur in the context of a SQL Server-initiated **dump**. Users who attempt to programmatically duplicate one of these RPCs are prevented from doing so.
3.51.2	Database device #%1!, name %2! has already been declared as #%3!, name %4!.	Internal error, contact Sybase Technical Support. The **dump** or **load** command as translated by SQL Server does not make sense. The *sysdevices* table may be corrupt.
3.52.2	Database device #%1! has not been declared.	
3.53.2	Malformed option list received, RPC: %1!.	Internal error, contact Sybase Technical Support. The **dump** or **load** options as passed from SQL Server do not make sense. The problem may be possible to work around by rearranging, adding or removing options to the **dump** or **load** command.
3.54.1	Waiting for processes to complete.	This message is sent to the session issuing a **shutdown** command if there are outstanding dump or **load** requests and **nowait** was not specified.
3.55.2	RPC %1!, parameter ‘%2!’: ‘%3!’ is an invalid value.	Internal error, contact Sybase Technical Support. SQL Server requested an unknown function.
4.1.2	Device ‘%1!’: volume ‘%2!’ appears to contain data written in a format that the Backup Server does not recognize. Please use a different volume, or initialize this volume through the operating system.	Digital OpenVMS only: the tape may be of foreign origin. Digital OpenVMS volume init may know how to confirm that data can be overwritten. Will be followed by an additional operating system message elaborating on the situation.
4.2.3	SQL Server sent an overlapping disk piece.	Database is corrupt. Run **dbcc** diagnostics on your database and refer to *SQL Server Error Messages* for corrective measures.
4.3.2	Device %1! is not in use by any session on this server.	Indicates user addressed **sp_volchanged** execution to an inactive device. Check parameters.
4.4.2	Device %1! does not belong to session <%2!>. Notification failed.	Indicates a device specified in an **sp_volchanged** execution does not belong to the named session. Check parameters.
4.5.2	Notification attempt failed--message channel for session <%1!> is closed.	Indicates user addressed **sp_volchanged** execution to a nonexistent session. Check parameters.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.6.2	Failed to create or attach shared tape I/O buffers.	Indicates unavailability of shared memory on the platform, possibly caused by memory shortage or kernel configuration limit reached. Check resource availability. This is probably a Backup Server problem.
4.7.2	Device %1! already in use.	A session attempted to use an archive device owned by another session. Select a different archive device.
4.8.2	Mirror device types don't match: %1!.	This error is part of a not yet implemented feature, included here only for completeness.
4.9.2	Label validation error: read returned %1! reading label. Read:n%2!	These errors indicate a mislabeled or illegal-format archive volume. User should use offline commands to verify that the tape contains Sybase information. The following commands submitted repeatedly in order will display the beginning and ending label characters for each file on the tape: **dd if=<tape_device> bs=2048 | od -c** **mt -f <tape_device> fsf 1** **dd if=<tape_device> bs=2048 | od -c** Zero bytes reported from "dd" and "od" denotes no further data on the tape.
4.10.2	Label validation error: first label not VOL1.	Same as Error # 4.9.2
4.11.2	Label validation error: seek to %1!s failed, %2!.	Same as Error # 4.9.2
4.12.2	Label validation error: %1! label not found.	Same as Error # 4.9.2
4.14.2	Label validation error: too many header labels.	Errors similar to 4.9.2–4.12.2.
4.15.2	Label validation error: seek to trailer labels failed.	Errors similar to 4.9.2–4.12.2
4.16.2	Label validation error: read of tape mark failed.	Errors similar to 4.9.2–4.12.2
4.17.2	The stripes on the specified devices have completed, but more dump stripes exist.	This message arises when loading from fewer stripes than the number used at dump time. At load time, there must be at least one instance of every device type used at dump time.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.18.2	Unrecognized / Unsupported file type received - device name (‘%1!’).	The archive device type is either an operating system file, raw fixed disk, raw unloadable disk (for example, floppy disk), tape, or network location (*device_name AT remote_backup_server_name*). Select a device of a legal type.
4.19.2	Label validation error: too many trailer labels.	More errors similar to 4.9.2–4.12.2.
4.20.2	Label validation error: write of tape mark failed.	More errors similar to 4.9.2–4.12.2.
4.21.2	Label validation error: write of ending tape mark failed.	More errors similar to 4.9.2–4.12.2.
4.22.1	Device %1! option %2! not valid.	Indicates a device option not appropriate for the device type. Review legal options for device types.
4.23.2	%1!: read %2!: %3!.	Indicates a read or write system call error on a disk or file device. Investigate possible causes (for example, out-of-space condition.) This error aborts the **dump** or **load**.
4.24.2	%1!: write %2!: %3!.	Same as Error # 4.23.2
4.25.2	Device ‘%1!’: %2! is an inappropriate density value for this device.	Digital OpenVMS only: the density value specified in the **dump** or **load** command is not valid. Expected values are: *none specified*, 800, 1600, 6250, 6666, 10000, and 38000.
4.26.2	Volume validation error: failed to obtain device information, device: %1! error: %2!.	Backup Server is unable to sense the type of a given device. Verify that the device is of a legal type and that the Backup Server process has operating system permission to access it.
4.27.2	Volume validation error: attempt to close %1! returned %2!.	A UNIX file failed to close. The first parameter is the file name and the second is the UNIX error message that was returned.
4.28.2	Device ‘%1!’: set mode failed on channel %2!; status = 0x%3!.	Digital OpenVMS only: could not set device density. The “status” line will contain the operating system error code.
4.29.2	Device ‘%1!’: sense mode failed on channel %2!; status = 0x%3!.	Digital OpenVMS only: could not sense device characteristics. The “status” line will contain the operating system error code.
4.30.2	Volume validation error: attempt to open %1! returned %2!.	Indicates an error opening or closing a database device.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.31.2	Device '%1!': failed to skip back to beginning of tape on channel %2!; status = 0x%3!.	Digital OpenVMS only: volume init error. The "status" line will contain the operating system error code.
4.33.2	Header labels of rejected volume:	Display of the data Backup Server found where it expected ANSI volume labels. Verify that the proper volume is loaded.
4.39.1	Device '%1!': illegible file date & time found; continuing.	This error is raised when Backup Server attempts to list the contents of a tape. Media failure or corruption could be indicated. Backup Server will continue listing contents (other files may be fine).
4.41.1	Creating new disk file %1!.	This message occurs when attempting to **dump** to a non-existent disk *FNAME*.
4.42.2	Couldn't obtain channel to multibuffering subprocess, error: %1!	An attempt to fork a subprocess failed.
4.43.2	Couldn't create multibuffering subprocess.	Backup Server was unable to complete start-up of the multibuffering subprocess. File descriptors may be exhausted (Error 4.42). Retry session when Backup Server activity is lower. Possible virtual memory shortage (Error 4.43).
4.45.2	The maximum number of %1! stripe devices has been exceeded.	A single dump or load session may use a maximum of 32 archive stripes. Reduce the number of archive devices in the **dump** or **load** command.
4.46.2	Length error on I/O -- transferred %1! bytes, expecting to transfer %2! bytes.	Digital OpenVMS only: Backup Server requested the operating system to read or write %2 bytes and only %1 bytes were actually transferred. Probably device or media failure.
4.51.2	Archive devices '%1!' and '%2!' do not belong to the same file group.	Internal error, contact Sybase Technical Support. The tapes/devices do not belong to the same archive group. To the Backup Server, it appears that "tapes" from two different dumps are mounted in devices requested by the current **load**.
4.52.2	There should be %1! load stripes, but the command only specifies %2!. One or more required devices have been omitted from the command line.	Backup Server supports database loads on fewer devices than were used for the dump. However, at least one device of each type (4 mm, 8 mm, and so on) must be supplied. Backup Server verifies that a **load** command specifies all needed device types. If it does not, this error is raised.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.53.2	The **load** command specifies too many devices of type '%1!': expected %2!, got %3!.	Similar to 4.52.2; however, this is the reverse case. Backup Server cannot load from more stripes than were used for the original dump.
4.54.2	The **load** command specifies too few devices of type '%1!': expected %2!, got %3!.	At least one type of each device must be specified in the **load** command
4.55.2	Device validation error: couldn't open raw device %1!, error: %2!	These errors denote failure to obtain hardware characteristics after establishing the device type. Check permissions on the "device special" file, investigate the returned operating system error message.
4.56.2	Device validation error: couldn't obtain tape drive characteristics for device %1!, error: %2!	Same as Error # 4.55.2
4.57.2	Device validation error: couldn't obtain disk drive characteristics for device %1!, error: %2!	Same as Error # 4.55.2
4.58.1	Database %1!: %2! allocated kilobytes %3!ed.	This message conveys the progress of the dump or load session. %3 is "DUMP" or "LOAD".
4.59.3	Archive device %1! must either be a STRIPE or MIRROR.	Internal error, contact Sybase Technical Support. This error can occur only if SQL Server incorrectly translated a **dump** command.
4.62.3	Multibuffering subprocesses died, archive %1!.	This error message may indicate that the Backup Server cannot find the **sybmultbuf** program. Check *$SYBASE/bin* to see if **sybmultbuf** is there. This can also happen if the Backup Server cannot locate the **sybmultbuf** binary because it has received an incorrect parameter. The correct use of the **-M** flag is: **$SYBASE/bin/backupserver -M$SYBASE/bin/sybmultbuf** A third possibility is that the Backup Server has had trouble forking the **sybmultbuf** process. Check the Backup Server's *srv.log*, which may contain more information regarding this problem. For emulator failure accompanied by this error message, see "Errors in Backup Server Error Log" on page 3-18.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.63.2	End-of-volume reading labels, archive %**1!**.	Backup Server encountered a tape file mark when attempting to read the ANSI labels. Indicates validation attempt on a blank tape (not an error), a non-ANSI tape or programming error. The tape volume should be examined off line with the commands given above. If the volume was the final volume of the **load** and the I/O has proceeded normally to this point (as evinced by progress messages, etc.) then the **load** I/O can be considered successful. If not, the **load** should be reattempted.
4.64.3	SANITY: Scheduled ACK events exceed stripe count.	Internal error, contact Sybase Technical Support.
4.67.2	Device %1!: This tape device requires the CAPACITY option.	Backup Server does not know how to detect end-of-tape on this device. Therefore **capacity** must be specified on **dump**, or you must specify the device as a logical device name from *sysdevices*.
4.68.2	Write data to a network device (slave site: %1!, device: %2!) failed.	These messages are self-explanatory. Further details will appear in a 4.82.2 message, which will immediately follow this message.
4.69.2	Read data from the network device (slave site: %1!, device: %2!) failed.	Same as Error # 4.68.2
4.70.2	Device %1!: Unable to query ODM database for device attributes. ODM error code = %2!.	RS6000 only: Since Backup Server has failed to determine device characteristics, it cannot ensure that the device is configured for extended file marks.
4.71.2	Device %1!: Is not configured for extended file marks.	RS6000 only: Multi-file devices must be reconfigured with extended tape marks. This is so that tape marks can be overwritten on multi-file dumps.
4.74.2	Unable to %1! database information file %2! : %3!.	UNIX Backup Servers use a temporary file (created in */tmp*) to pass database device names to the **sybmultbuf** program. An I/O error with this file has occurred. User should verify that */tmp* is world-writable, has sufficient space, and contains no unused files of the form *BS_servername.number.*

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.75.3	Device %1!: %2! is not a legal virtual disk number.	Internal error, contact Sybase Technical Support. Backup Server cannot determine what portions of the specified database device should be backed up because information about the device sent from SQL Server does not correspond with the actual physical device.
4.76.3	Device %1!: %2! is not a legal virtual block number.	Same as Error # 4.75.3
4.77.2	Device %1! may not be used: minimum blocksize for I/O exceeds device maximum of %2!.	Backup Server requires tape devices to support an I/O size of at least 2K. The tape device found has a maximum I/O size less than 2K. This message should never appear. If it does, user should check device name specification. If correct, user must use another device.
4.78.2	Option %1!: illegal value %2!.	This indicates the user has supplied an illegal value for a device option. Review the device options and legal values.
4.79.2	Server %1!, device %2!: illegal I/O size %3!--max %4!.	Indicates a programming bug in a local or remote Backup Server. The I/O service layer has received a read or write request size that exceeds the available buffer size.
4.80.2	Server %1!, device %2!: You must use the no-rewind-on-close tape device.	Backup Server requires all tape devices to use the **no-rewind-on-close** option, so that it can fully control tape positioning.
4.81.2	Server %1!: device %2! is open for writing; you may not specify the @fname parameter when changing volumes.	The *@fname* parameter is meaningful only at load time, to select a certain file to load from a multi-file volume. At dump time, the Backup Server assigns a name to the file that will contain the dump. Message 6.28.1 returns the assigned name.
4.82.2	Operating system error, server %1! device %2!: code %3! message %4!.	This message appears when a system call (perhaps performed as part of a higher-level operation) has failed. It is intended primarily for diagnostic use and should be interpreted in the context of immediately preceding messages.
4.83.2	Device %1!: The specified blocksize %2! is not within the range of %3! to %4!.	The user-specified block size is out of range. Respecify block size within the listed limits.
4.84.1	Device %1!: The specified blocksize of %2! will be truncated to %3! which is a multiple of %4!.	The user-specified block size was truncated to be a multiple of 2048.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
4.85.1	Device %1!: Disk model %2! does not exist in the /etc/disktab file. Using default values for this device.	The disk device is not configured in */etc/disktab*, so a default disk block size will be used.
4.86.2	Unable to use asynchronous IO on the database device %1!. Backup Server must use asynchronous IO on a block special device because SQL Server is doing the same. Refer to the server release notes.	This is an HP-specific error. If SQL Server is using asynchronous I/O for a block special device, then Backup Server must do the same in order to guarantee disk coherency. Block I/O without asynchronous is a buffered I/O operation. With asynchronous I/O, it is not buffered. If two processes access the same disk, one using buffered I/O and the other using nonbuffered I/O, the current image of a disk page may be ignored by one of the processes.
4.87.2	Device %1!: uname() failed with error: %2!.	HP only: Backup Server is trying to identify the machine as an HP 800 or HP 700. The **uname** call (which gets this information) failed.
4.88.2	Device %1!: Could not determine device type due to unknown machine model. Expected models are HP9000/800 & 700 series.	Related to Error 4.87.2. After getting the machine name with the **uname** call, Backup Server could not identify the device type because it could not identify the machine type. Only 800 and 700 series machines are expected.
5.1.2	The mirror devices (site: %1!) and the primary device(site: %2!) must reside at the same site.	This message is part of the uncompleted dump mirroring feature. All mirrors for a stripe must reside on the same host.
5.2.2	Cannot allocate the login record.	This error occurs when the **dblogin** call fails, probably due to a memory resource problem.
5.3.2	Cannot open a connection to the slave site ‘%1!’.	An attempt to connect to a remote Backup Server encountered this failure. Look for: • Memory or swap space shortage • No remote Backup Server running • Wrong listening address
5.4.2	RPC (‘%1!’) initialization fails.	Internal error, contact Sybase Technical Support. Failure in **dbrpcinit**.
5.5.2	Cannot add a parameter to the RPC (‘%1!’) call.	Internal error, contact Sybase Technical Support. Failure in **dbrpcparam**.
5.6.2	Cannot send the RPC (‘%1!’).	Check to see that the local or remote Backup Server is still available.
5.7.2	RPC (‘%1!’) execution failed.	These messages point to difficulties in issuing an RPC to a remote Backup Server. Probable cause is an Open Server programming error.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
5.11.2	RPC bs_end_load execution failed - slave server: %1!, device: %2!.	These messages indicate failure to deliver the named RPCs to the remote Backup Server. Adjacent messages provide more detail. Probable causes are sudden death of the remote Backup Server, or a programming error.
5.12.2	Missing device name parameter @devname, RPC: %1!, command: %2!.	Internal error, contact Sybase Technical Support. Check the compatibility of local and remote Backup Servers.
5.13.2	Received an unknown command. RPC: %1!, command: %2!.	Same as Error # 5.12.2
5.14.2	Cannot set the value of an RPC return parameters, RPC: %1!	Same as Error # 5.12.2
5.15.2	Slave server received %1! bytes of data that is more than @count %2! specified in as_pagerun.	Internal error, contact Sybase Technical Support. Violation of the RPC API.
5.16.2	DB-Library error, error number %1!, severity %2!: %3!	The installed DB-Library error handling function returns this error. Adjacent messages provide more detail.
5.17.2	Cannot get the character set currently in use.	The remote Backup Server could not set the character set to its default setting for the initial connection from a client of the local Backup Server.
5.18.2	Cannot set character set '%1!' in the login record.	The remote Backup Server could not set the character set to the set specified by the remote connection.
5.19.2	Cannot get the language currently in use.	The remote Backup Server could not set the language to its default setting for the initial connection from a client of the local Backup Server.
5.20.2	Cannot set language '%1!' in the login record.	The remote Backup Server could not set the language specified by the remote connection.
6.28.1	Dumpfile name '%1!' section number %2! mounted on %3! '%4!'	This message has a critical role. It contains the file name assigned to the archive file at dump time. If the dump goes to a multifile volume, the user or the application must record the dumpfile name in persistent storage and specify it at load time in order to locate the file for loading from the multifile archive. If the dumpfile name is lost, it will be necessary to scan the tape off-line with the commands given under message 4.12.2 to locate the dumpfile name.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
6.30.1	Device %1!: Volume cataloguing complete.	This is an informational message indicating that the **load with listonly** command has completed successfully for the specified archive device.
6.31.2	Volume rejected.	Backup Server refuses to use the mounted volume. Adjacent messages provide more detail. The volume either has non-ANSI structure or there is a Backup Server programming error.
6.32.2	%1!: volume not valid or not requested (server: %2!, session id: %3!.)	The data on the device is not in proper dump format. If you are loading, use another volume. If you are dumping, user may overwrite.
6.33.2	%1!: Volume already contains %2! kilobytes, which is larger than the specified capacity of %3! kilobytes.	A multi-file dump device contains more data than was specified in the **capacity** clause.
6.35.2	Volume validation error: bad magic number %1!, expected %2!.	Indicates a non-Sybase archive mounted on the drive. Replace with a Sybase dump archive.
6.36.2	Header labels of rejected volume:	Preamble message to display labels of rejected volume.
6.37.2	Volume validation error: Load block size of %1! must equal volume block size of %2!.	Indicates a non-Sybase archive (illegal block size) mounted on the drive. Replace with a Sybase dump archive.
6.41.1	Header labels of mounted volume:	Preamble message to display labels of rejected volume.
6.45.1	Be sure to remove the tape / floppy from drive %1! (server: %2!, session id: %3!).	Reminder message appears when **nounload** (default) is specified for an unloadable device.
6.46.1	%1!: Mount volume %2!.	Requests mounting of the next volume in sequence for **load**.
6.47.1	%1!: Mount the next volume to write.	Requests mounting of the next volume in sequence to continue **dump**.
6.48.1	%1!: Volume on device ‘%2!’ has restricted access (code %3!).	Reports ANSI restricted-access code in label prior to confirming overwrite request.
6.49.1	%1!: Volume to be overwritten on ‘%2!’ has not expired: creation date on this volume is %3!, expiration date is %4!.	Requests verification of overwrite of unexpired volume.
6.50.1	%1!: Dumpfile ‘%2!’ section %3! found instead of ‘%4!’ section %1!.	The volume just read is out of order. Mount the correct volume.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
6.51.1	%1!: Mount the next volume to search.	Reports that the requested dumpfile (specified by the **file** option) was not found on the volume.
6.52.1	%1!: Volume to be overwritten on '%2!' has unrecognized label data.	A single file dump device has non-Sybase dump data. This is a prompt set to the user asking if the data is to be overwritten.
6.53.1	%1!: Volume on device '%2!' cannot be opened for write access. Mount another volume.	Check write protection.
6.54.1	%1!: Volume on device '%2!' is expired and will be overwritten.	A single dump file device has media with expired dump data on it. The user is prompted whether the data is to be overwritten.
6.55.1	The volume mounted on '%1!' does not belong to the same archive as other, previously mounted volumes.	The creation time of this volume does not match the other volumes or stripes.
6.61.2	Volume validation error: Volume change request not allowed on non-mountable devices, device name: %1!.	Requests mounting of next volume in sequence to locate requested dump file.
6.65.2	Volume validation error: illegal volume change, device %1!: volume for stripe %2! mounted while stripe %3! loading still in progress.	Reminds user that all volumes of a dump-time stripe must complete loading on the device before the first volume of another dump-time stripe may begin loading.
6.66.2	Volume validation error: illegal volume change, device %1!: volume mounted out of order, expected volume %2!, got volume %3!.	Volumes must be loaded in the same order in which they were dumped.
6.78.1	EXECUTE sp_volchanged\n\t@session_id = %1!,\n\t@devname = '%2!%3!',\n\t@action = { '%4!' | '%5!' | '%6!' }%7!	Template for **sp_volchanged** execution. **proceed** means continue operation with the currently mounted volume. **retry** means redo the validation check that led to the prompt (presumably on a new volume). **abort** means abort the **entire** dump or load session.
6.80.2	Session %1! is exiting by request from sp_volchanged. Data written in this session are incomplete and invalid. Aborting session.	The user requested an abort through **sp_volchanged**.
6.81.2	Unrecognized volume change prompt id (internal error); task must exit.	Indicates a Backup Server programming error. Retry the **dump** or **load** command, avoiding unnecessary volume manipulations.
6.82.2	%1! is not a recognized sp_volchanged action.	User provided an illegal *@action* value to **sp_volchanged**. The value must be **proceed**, **retry**, or **abort**.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
6.83.2	Volume validation error: Volume name mismatch, volume name %1!, expected volume name %2!.	Volumes within a stripe must have the same name in the ANSI VOL1 label. It is possible for a user to execute the **dump** command properly and still receive this error at load time, if the first volume was overwritten using the **dumpvolume** option and the second volume, with a different name, was appended to.
6.84.2	Volume validation error: illegal volume change, device %1!: stripe %2! is already loaded.	User remounted a tape for a stripe that has already been loaded.
7.1.2	Memory error: failed to allocate an \”%1!\” structure.	Not enough memory available to Backup Server. See “Note” at the end of this chapter.
7.2.2	Option \”%1!\”: %2! is an invalid value; value must be an even multiple of %3!.	Illegal value for a **dump** or **load** option.
7.3.2	Option error: when you specify %1!, you may not specify %2!.	Illegal combination of options in a **dump** or **load** command.
7.4.2	Option \”%1!\”: \”%2!\” is an invalid value -- check the documentation for allowable values.	Unrecognized value specified for a **dump** or **load** command.
7.5.2	\”%1!\” is a badly formed value string.	Syntax error in **dump** or **load** command option.
7.6.2	Option \”%1!\”: \”%2!\” is an invalid volume label (too long).	Volume label is longer than 6 characters.
7.7.2	Option \”%1!\” is not valid.	Unrecognized **dump** or **load** option.
7.8.2	Option \”%1!\” may not be negated.	You cannot use **no** with this option.
7.9.2	Option \”%1!\” does not take any value; remove ‘=’.	This option does not allow you to specify a value.
7.10.2	Option \”%1!\” may not have more than one value.	User specified more than one value after “=”, and the option only permits one value.
7.11.2	Option \”%1!\” may not take a value when it is negated.	User specified an option with a value and **no**. The combination is illegal for this option.
7.12.2	Option \”%1!\” requires a value, but none was specified.	You must specify a value for these options: **density, blocksize, capacity, dumpvolume, retaindays**, or **file**.
7.13.2	Option \”%1!\”: \”%2!\” is not unique -- supply more characters.	User abbreviated a keyword following a value, and the abbreviation is ambiguous.
7.14.2	Option error: when you specify %1!, you must also specify %2!.	Invalid combination of options.

Table 3-4: Backup Server error messages (continued)

Error #	Message	Explanation
7.15.2	Option \”%1!\” is not valid for the present command. Please check the documentation for correct usage.	**load** option used during a **dump** command, or vice versa.

➤ ***Note***

Any “shortage of memory” message may indicate that both swap and physical memory are in short supply. Increasing either may avert this error. Make sure that you have enough physical memory so that SQL Server and Backup Server can run without being swapped.

4 System Database Recovery

This chapter provides step-by-step procedures for recovering from various disaster situations involving SYBASE system databases or the entire master device.

◆ *WARNING!*

Storing user databases on the master device is not recommended, as this greatly complicates disaster recovery.

Ensuring Recoverability

The best time to prepare for a disaster is before it happens.

Review the procedures in this chapter before an actual disaster occurs, such as a power failure, hard disk crash, or other severe problem that could cause the loss of your master device, your *master* database, or other vital system resource. Here are some helpful hints for making these procedures the most effective:

- Create complete, detailed scripts to recreate your system exactly as it existed before the disaster and perform recovery as efficiently as possible. In particular, your scripts should contain the following information:
  - Copies of key system tables in the *master* database, particularly *sysdatabases*, *sysdevices*, and *sysusages*.
  - Records of all changes made to *syslogins* and *sysloginroles*. You may want to keep an ongoing script of all the **sp_addlogin** and **sp_droplogin** commands.
  - Records of all changes from the default made by **sp_configure**, especially any changes to the **devices**, **locks** or **memory** configuration parameters.
  - Records of creations and modifications of system and user databases, particularly for structural changes, and particularly for *master*.
  - SQL records. Even if you are adding only a single disk device or a couple of logins, it is good system administration practice to save all this information in scripts and hard copy.

- Back up (using **dump database**) the *master* database frequently to help simplify solving problems with the *master* database. Back it up after any changes to system tables, especially *sysusages, sysdatabases, sysdevices*, and *syslogins*.
- Truncate the *master* database log frequently.
- Truncate the *model* database log periodically for SQL Server release 4.2.
- Keep statistics on how much time and space are required for dumps and loads.
- Avoid keeping user databases on the master device, as it complicates recovery scenarios.
- Always issue a **dump database** command after the following:
  - **bcp** (fast version)
  - **create index**
  - **select into**
  - **dump transaction with no_log**
  - **dump transaction with truncate_only**
- Where appropriate, automate the use of operating system threshold procedures and scripts that run backups.
- Verify that your *interfaces* file is correct.
- Catalog and label your backup media carefully.
- Try to run **dbcc** commands at the time you make dumps to ensure that the dump is not corrupted.

Consult the *System Administration Guide* and the *SQL Server Reference Manual* to learn more about the procedures described in this chapter so that you are ready for an emergency.

Master Device Completely Lost

Perform the following procedure to recover a SQL Server after completely losing a master device. Use this procedure only if your SQL Server was installed with the default sort order.

If you have installed a **non-default sort order**, see "Valid Dump with Non-Default Sort Order" on page 4-5.

The procedure for recovering a master device is different for release 10.0 and later than it is for pre-10.0 SQL Servers. Follow the directions very carefully to ensure that you use the proper instructions.

If a Valid Dump of the *master* Database Exists

If your SQL Server is release 10.0 or later, see "For 10.0 and Later SQL Server" on page 4-4.

For Pre-10.0 SQL Server

1. Build a new master device. See "How to Build a New Master Device" on page 6-1 for instructions.
2. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.

➤ ***Note***

If the *master* database was dumped to one of the default dump devices provided by SQL Server, you can skip steps 3 and 4, and proceed to step 5.

3. Execute the **installmaster** script. See "How to Run the installmaster/installmodel Scripts" on page 6-4.
4. Add the SYBASE dump device for loading the *master* database. See "How to Add a SYBASE Dump Device" on page 6-8.
5. Load the *master* database from backup. See "How to Load the master Database from Backup" on page 6-5.

   SQL Server shuts itself down after the load is complete.
6. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See "How to Alter the devices Parameter Manually" on page 6-7.
7. Start SQL Server.

8. Execute the **reconfigure** command. See “How to Execute the reconfigure Command” on page 6-8.
9. Restore system catalog information for the *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.
10. Load or rebuild the *model* database if necessary. See “How to Alter the model Database” on page 6-8.
11. Dump the *master* database if step 9 was needed.
12. Drop, re-create, and load any user databases located fully or partially on the master device.

For 10.0 and Later SQL Server

1. Build a new master device. See “How to Build a New Master Device” on page 6-1 for instructions.
2. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
3. Ensure that the SQL Server has the correct name for the Backup Server in the *sysservers* table. Refer to “How to Set SYB_BACKUP Manually in SQL Server” on page 6-10 for instructions.
4. Load the *master* database from backup using the **load database** command to specify the physical device or file name to reference. For example:

   ```
   1> load database master from device_name
   2> go
   ```

   SQL Server will shut itself down after the load is complete.
5. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See “How to Alter the devices Parameter Manually” on page 6-7.
6. Start SQL Server.
7. Execute the **reconfigure** command. See “How to Execute the reconfigure Command” on page 6-8.
8. Restore system catalog information for the *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.

9. Load or rebuild the *model* database if necessary. See "How to Alter the model Database" on page 6-8.
10. Drop, re-create, and load any user databases located fully or partially on the master device.

◆ ***WARNING!***

Storing user databases on the master device is not recommended, as this greatly complicates disaster recovery.

Valid Dump with Non-Default Sort Order

If your SQL Server is release 10.0 or later, go to "10.0 and Later SQL Server" on page 4-6.

Pre-10.0 SQL Server

1. Comment out the entry for the SQL Server in the interfaces file.
2. Rename the *RUN_SERVER* file for the SQL Server to *RUN_SERVER.old*.
3. Run **sybconfig** and build a new master device with your non-default sort order and character set. This creates a new entry in the interfaces file as well as a new *RUN_SERVER* file to replace the one you renamed in step 2.
4. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.

➤ ***Note***

If the *master* database was dumped to one of the default dump devices provided by SQL Server, you can skip steps 5, 6, and 7, and proceed to step 8.

5. Execute the **installmaster** script. See "How to Run the installmaster / installmodel Scripts" on page 6-4.
6. Add the SYBASE dump device for loading the *master* database. See "How to Add a SYBASE Dump Device" on page 6-8.
7. Load the *master* database from backup. See "How to Load the master Database from Backup" on page 6-5.

   SQL Server shuts itself down after the load is complete.

8. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See “How to Alter the devices Parameter Manually” on page 6-7.
9. Start SQL Server.
10. Execute the **reconfigure** command. See “How to Execute the reconfigure Command” on page 6-8.
11. Restore system catalog information for the *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.
12. Load or rebuild the *model* database if necessary. See “How to Alter the model Database” on page 6-8.
13. Dump the *master* database if step 11 was needed.
14. Drop, re-create, and load any user databases located fully or partially on the master device.

10.0 and Later SQL Server

1. Comment out the entry for the SQL Server in the interfaces file.
2. Rename the *RUN_SERVER* file for the SQL Server to *RUN_SERVER.old*.
3. Run **sybinit** and build a new master device with your non-default sort order and character set. This creates a new entry in the interfaces file as well as a new *RUN_SERVER* file to replace the one you renamed in step 2.
4. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
5. Ensure that the SQL Server has the correct name for the Backup Server in the *sysservers* table. Refer to “How to Set SYB_BACKUP Manually in SQL Server” on page 6-10 for instructions.
6. Load the *master* database from backup using the **load database** command to specify the physical device or file name to reference. For example:

   ```
   1> load database master from device_name
   2> go
   ```

   SQL Server shuts itself down after the load is complete.

7. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See "How to Alter the devices Parameter Manually" on page 6-7.
8. Start SQL Server.
9. Execute the **reconfigure** command. See "How to Execute the reconfigure Command" on page 6-8.
10. Restore system catalog information for the *master* database if changes were made to it since the last dump. See "How to Restore System Table Information in master Database" on page 6-5.
11. Load or rebuild the *model* database if necessary. See "How to Alter the model Database" on page 6-8.
12. Drop, re-create, and load any user databases located fully or partially on the master device.

If a Valid Dump of the *master* Database Does Not Exist

If your SQL Server is release 10.0 or later, see "For 10.0 and Later SQL Server" on page 4-8.

For Pre-10.0 SQL Server

1. Build a new master device. See "How to Build a New Master Device" on page 6-1.
2. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.
3. Execute the **installmaster** and the **installmodel** scripts. See "How to Run the installmaster / installmodel Scripts" on page 6-4.
4. Execute **sp_configure** to restore the original value of the **devices** configuration parameter. See the *SQL Server Reference Manual* for instructions. Execute **reconfigure** to initialize the changes.
5. Restore system catalog information for the *master* database. See "How to Restore System Table Information in master Database" on page 6-5.
6. Alter the *tempdb* database if necessary. See "How to Alter tempdb" on page 6-7.
7. Alter the *model* database if necessary. See "How to Alter the model Database" on page 6-8.
8. Dump the *master* database.

9. Drop, re-create, and load any user databases located fully or partially on the master device.
10. Shut down SQL Server.
11. Start SQL Server in multi-user mode.

For 10.0 and Later SQL Server

1. Build a new master device. See "How to Build a New Master Device" on page 6-1.
2. Reset the **devices** configuration parameter manually. Refer to "How to Alter the devices Parameter Manually" on page 6-7 for instructions.
3. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.
4. Restore system catalog information for the *master* database. See "How to Restore System Table Information in master Database" on page 6-5.
5. Alter the *tempdb* database if necessary. See "How to Alter tempdb" on page 6-7.
6. Alter the *model* database if necessary. See "How to Alter the model Database" on page 6-8.
7. Execute the **installmaster** and the **installmodel** scripts. See "How to Run the installmaster / installmodel Scripts" on page 6-4.
8. Use **sp_addserver** to add a SYB_BACKUP entry to the *sysservers* table:

```
1> sp_addserver "SYB_BACUKP", null,
2> <correct backup server name>
3> go
```

9. Dump the *master* database.
10. Drop, re-create, and load any user databases located fully or partially on the master device.
11. Shut down SQL Server.
12. Start SQL Server in multi-user mode.

◆ ***WARNING!***

Storing user databases on the master device is not recommended, as this greatly complicates disaster recovery.

Master Database Corrupt and SQL Server Does Not Start

Perform these steps to recover a *master* database that is corrupt and unusable by SQL Server. These procedures assume that the rest of the master device is intact.

The procedure for recovering a *master* database is different for release 10.0 and later than it is for pre-10.0 SQL Servers. Follow the directions very carefully to ensure that you use the proper instructions.

If a Valid Dump of the *master* Database Exists

If your SQL Server is release 10.0 or later, see "For 10.0 and Later SQL Server" on page 4-10.

For Pre-10.0 SQL Server

1. Rebuild the *master* database without initializing the master device. See "How to Rebuild master Database and Leave Master Device Intact" on page 6-2.
2. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.

➤ ***Note***

If the *master* database was dumped to one of the default dump devices provided by SQL Server, you can skip steps 3 and 4 and proceed to step 5.

3. Execute the **installmaster** script. See "How to Run the installmaster/installmodel Scripts" on page 6-4.
4. Add the SYBASE dump device for loading the *master* database. See "How to Add a SYBASE Dump Device" on page 6-8.
5. Load the *master* database from backup. See "How to Load the master Database from Backup" on page 6-5.

   SQL Server shuts itself down after the load is complete.
6. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See "How to Alter the devices Parameter Manually" on page 6-7.
7. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.

8. Execute the **reconfigure** command. See “How to Execute the reconfigure Command” on page 6-8.
9. Restore system catalog information for the *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.
10. Dump the *master* database.
11. Shut down SQL Server.
12. Start SQL Server in multi-user mode.

For 10.0 and Later SQL Server

1. Rebuild the *master* database without initializing the master device. See “How to Rebuild master Database and Leave Master Device Intact” on page 6-2.
2. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
3. Ensure that the SQL Server has the correct name for the Backup Server in the *sysservers* table. Refer to “How to Set SYB_BACKUP Manually in SQL Server” on page 6-10 for instructions.
4. Load the *master* database from backup using the **load database** command to specify the physical device or file name to reference. For example:

   ```
   1> load database master from "device_name"
   2> go
   ```

   SQL Server shuts itself down after the load is complete.
5. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See “How to Alter the devices Parameter Manually” on page 6-7.
6. Start SQL Server in single-user mode.
7. Execute the **reconfigure** command. See “How to Execute the reconfigure Command” on page 6-8.
8. Restore system catalog information for the *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.
9. Execute the **installmaster** and **installmodel** scripts. See “How to Run the installmaster/installmodel Scripts” on page 6-4.

10. Dump the *master* database.
11. Shut down SQL Server.
12. Start SQL Server in multi-user mode.

If a Valid Dump of the *master* Database Does Not Exist

If your SQL Server is release 10.0 or later, see “For 10.0 and Later SQL Server” on page 4-11.

For Pre-10.0 SQL Server

1. Rebuild the *master* database without initializing the master device. See “How to Rebuild master Database and Leave Master Device Intact” on page 6-2.
2. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
3. Execute the **installmaster** script. See “How to Run the installmaster / installmodel Scripts” on page 6-4.
4. Execute **sp_configure** to restore the original values of the devices. See the *SQL Server Reference Manual* for instructions. Execute **reconfigure** to initialize the changes.
5. Restore system catalog information for the *master* database. See “How to Restore System Table Information in master Database” on page 6-5.
6. Dump the *master* database.
7. Shut down SQL Server.
8. Start SQL Server in multi-user mode.

For 10.0 and Later SQL Server

1. Rebuild the *master* database without initializing the master device. See “How to Rebuild master Database and Leave Master Device Intact” on page 6-2.
2. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
3. Execute the **installmaster** script. See “How to Run the installmaster / installmodel Scripts” on page 6-4.

4. Execute **sp_configure** to restore the original values of the devices. See the *SQL Server Reference Manual* for instructions. Execute **reconfigure** to initialize the changes.
5. Restore the system tables information contained in the *master* database. This information describes all SYBASE devices and user databases. Refer to “Restoring Device and Database Information in the System Catalog” on page 6-5 for instructions.
6. Use **sp_addserver** to add a SYB_BACKUP entry to the *sysservers* table:

```
1> sp_addserver "SYB_BACUKP", null,
2> <correct backup server name>
3> go
```

7. Dump the *master* database.
8. Shut down SQL Server.
9. Start SQL Server in multi-user mode.

master Database is Corrupt but SQL Server Starts

Perform these steps to recover a *master* database that is corrupt but usable by SQL Server. For example, some tables in the *master* database are corrupt but SQL Server can boot and the System Administrator can use the *master* database to a certain extent. This procedure assumes that the rest of the master device is intact and that you have a valid dump of the *master* database.

If a Valid Dump of the *master* Database Exists

1. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
2. Load the *master* database from backup. See “How to Load the master Database from Backup” on page 6-5.

   SQL Server will shut itself down after the load is complete.
3. Start SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3.
4. Restore system catalog information of *master* database if changes were made to it since the last dump. See “How to Restore System Table Information in master Database” on page 6-5.
5. Start SQL Server in multi-user mode.

If a Valid Dump of the *master* Database Does Not Exist

If you do not have a valid dump of the *master* database, you have lost your *master* database. (See “Master Database Corrupt and SQL Server Does Not Start” on page 4-9.) However, you can print system catalog information that will be helpful in restoring the *master* database: s*ysusages*, *sysdatabases*, *sysdevices*, and *syslogins*. You can do this with the **select * from** command. See the *SQL Server Reference Manual* for complete instructions.

Lost or Corrupted *model* Database

If you can use the *model* database with the **use model** command, and if you have a valid dump of the database, then you can load the *model* database from backup.

If you cannot use the *model* database or do not have a dump of the *model* database, follow the appropriate set of steps below.

➤ **Note**

If you have not changed *model* at all, you can replace the procedure on this page with a simpler one: use **sybinit** to build a "dummy" SQL Server, dump *model* from the dummy to your SQL Server, and then delete the dummy SQL Server.

The procedure for recovering a *model* database is different for release 4.9.1 and later than it is for pre-4.9.1 SQL Servers. Follow the directions very carefully to ensure that your use the proper instructions.

If your SQL Server is release 4.9.1 or later, see "For 4.91 and Later SQL Server" on page 4-15.

For Pre-4.9.1 SQL Server

1. Run **dbcc checkalloc** and **dbcc checkdb** on the *master* database. If there are no errors, proceed to the next step. If you encounter errors, see *SQL Server Error Messages* or contact Sybase Technical Support.
2. Dump the *master* database. Save the contents of *sysusages*, *sysdatabases*, *syslogins*, and *sysdevices* in case problems occur.
3. Dump any user databases located fully or partially on the master device.
4. Build a new master device. See "How to Build a New Master Device" on page 6-1.

◆ ***WARNING!***

Do not use *installmodel* in place of this procedure.

5. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.

6. Execute the **installmaster** and the **installmodel** scripts. See "How to Run the installmaster / installmodel Scripts" on page 6-4.

➤ ***Note***

If the *master* database was dumped to one of the default dump devices provided by SQL Server, you can skip step 7 and proceed to step 8.

7. Add the SYBASE dump device for loading the *master* database. See "How to Add a SYBASE Dump Device" on page 6-8.
8. Load the *master* database from the backup you created in step 2. "How to Load the master Database from Backup" on page 6-5.

   SQL Server will shut itself down after the load is complete.
9. With SQL Server still down, manually reestablish the **devices** configuration parameter if necessary. See "How to Alter the devices Parameter Manually" on page 6-7.
10. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.
11. Execute the **reconfigure** command. See "How to Execute the reconfigure Command" on page 6-8.
12. Load any user databases (those dumped in step 3) onto the master device.
13. Rebuild the *model* database. See "How to Alter the model Database" on page 6-8.
14. Dump the *model* database.
15. Shut down SQL Server.
16. Start SQL Server in multi-user mode.

For 4.91 and Later SQL Server

1. Run **buildmaster -x**.
2. Reload any user-specific structures or data.

Loss of a Device Containing Pieces of *tempdb*

Follow this procedure if a device containing pieces of *tempdb*, other than the master device, has been lost:

1. Start SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3.
2. Print out the *sysusages* table for *tempdb*.
3. Delete all but the first entry in *sysusages* for *tempdb* (*dbid*=2). Make sure that the *segmap* column for this entry is 7. If the *model* database has been increased beyond its default size, do not reduce the size of *tempdb* so that it is smaller than *model*. Call Sybase Technical Support.

◆ ***WARNING!***

disk refit or disk reinit will fail on any *master* database on which this procedure is used.

For example:

```
1> begin transaction
2> delete master..sysusages
3> where dbid=2 and lstart != 0
4> go
```

```
1> update master..sysusages set segmap = 7
2> where dbid = 2
3> go
```

```
1> select * from master..sysusages where dbid=2
2> go
```

4. If the above **select** command produced the following output, continue to step 5:

```
dbid  segmap  lstart  size  vstart

2     7       0       1024  2564
```

   If it did not, roll back the transaction and contact Sybase Technical Support.

5. Commit the transaction (disallowing system catalog updates), and shut down SQL Server using the following commands:

```
1> commit transaction
2> go
```

```
1> shutdown
2> go
```

6. Start SQL Server in multi-user mode.
7. Drop (**sp_dropdevice**) and reinitialize (**disk init**) the lost device. If user databases are on the same device with *tempdb*, you may

have to drop those databases also, before dropping and reinitializing the lost device.

8. Use the **alter database** command to restore *tempdb* to the desired size.
9. Dump the *master* database.

Master Device Is Going Bad

1. Ensure the consistency of the *master* database by running **dbcc checkalloc** and **dbcc checkdb**.
2. Ensure the consistency of any user databases located fully or partially on the master device by running **dbcc checkalloc** and **dbcc checkdb**.
3. Dump any user databases located fully or partially on the master device. Save the contents of *sysusages*, *sysdevices*, *sysdatabases*, and *syslogins*.
4. If the consistency checks on the *master* database do not produce errors, and changes have been made since the last backup, dump the *master* database.
5. Perform steps 1 and 2 for the *model* database if it has been changed since the original installation.
6. Have your hardware checked and repaired. If the device is replaced, follow the steps above. See “Master Device Completely Lost” on page 4-3.

Cannot Start SQL Server After Altering Configuration

When SQL Server starts, it reads the configuration parameters contained on a portion of the master device called the **configuration block**. The values of these variables are used at start-up time to determine how much memory to allocate for various resources needed by SQL Server. If there is not enough memory available to satisfy all the requests, SQL Server will not start. This situation most often occurs when one or more erroneously high values are set with the **sp_configure** and **reconfigure** commands.

Refer to “How to Reset SQL Server to Its Default Configuration” on page 6-9 for information about resetting configuration parameters.

5 Stored Procedure Processing and Management

This chapter describes how SQL Server processes stored procedures. It also provides some of the background needed to manage stored procedures on your SQL Server. All the information in this section is true for all stored procedures and triggers.

Storing and Executing Procedures

When a stored procedure is created, the ASCII text of the procedure is stored in the *syscomments* table of the current database. In addition, a normalized form of the procedure, called a **query tree**, is stored in the *sysprocedures* table.

Building the Query Tree: Resolution

The process of building a query tree is called **resolution**. This process parses the SQL into a more efficient format and resolves all objects involved into their internal representations. For example, table names are resolved into their object IDs and column names are resolved into their column IDs.

Building the Query Plan: Compilation

The process of building a query plan is called **compilation**. SQL Server builds a query plan on the first execution of a stored procedure. When the procedure executes, SQL Server reads the corresponding query tree from the *sysprocedures* table and loads it into the procedure cache. SQL Server then creates a query plan and places it in the procedure cache. The query plan is the optimized data access path that SQL Server uses to execute the procedure.

SQL Server determines the optimal data access path and builds the query plan based on the following information:

- The SQL stored in the query tree
- Statistics for each table and index referenced in the procedure
- The values of any parameters passed to the procedure on the first execution

Since query plans are held only in procedure cache and not on disk, they must be rebuilt if the SQL Server was restarted since they last executed.

Multiple Users and the Query Plan

Stored procedures are reusable, not reentrant. This means that only one user at a time can execute a given copy of a procedure’s query plan. If two or more users try to execute the same procedure at the same time, SQL Server creates an additional query plan based on the parameters used in the latter execution. When a user finishes using the procedure, the query plan is available in cache for reuse by anyone with execute permissions.

If a second query plan for a stored procedure is generated, there is no guarantee that it is the same as the first one. If a user passes a different set of parameters to create the second invocation, the query plan may be different.

Also, a new query plan for a given procedure may be different from an older query plan if the user added an index to a referenced table or updated the statistics after the older query plan was generated.

Adding an index or updating statistics does not force recompilation, but the newly compiled query plan may be different from the old one, since it may take into account new or revised information. The additional query plans stay in procedure cache until they are swapped out and could cause one execution to run very differently from another, although the returned results are the same.

Resolving Execution Plan Differences

The database administrator or user has no control over or knowledge of which execution plan he will get for a given execution. This may explain unexpectedly different execution times for the same procedure given the same data and parameters. If this situation is suspected, dropping and re-creating the procedure will cause all existing plans to be flushed out of cache.

To ensure that you always get your own plan, you can use **exec with recompile** or **create with recompile**. Creating a procedure with the **with recompile** option decreases performance because every execution causes a compilation.

Procedure Recompilation

The process of building a query plan is called **compilation**. Recompilation is the creation of a new query plan from the existing query tree. Recompilation takes place whenever one of the following events occurs:

- The procedure is loaded from disk to the procedure cache.
- An index on any table referred to in the procedure is dropped.
- All copies of the execution plan in cache are currently in use and another user wants to execute the procedure.
- A procedure is executed using the **with recompile** option.
- A database administrator flags a table with the **sp_recompile** stored procedure. This causes SQL Server to re-resolve and then recompile any procedures or triggers that access that table, at execution time.

Dropping an index or a table referenced by a query causes SQL Server to mark the affected procedure as needing re-resolution and recompilation at execution time. Neither the **update statistics** nor the **create index** command causes an automatic recompilation of stored procedures.

Procedure Re-resolution

Like recompilation, re-resolution causes the generation of a new plan. In addition, re-resolution updates the existing query tree in the *sysprocedures* table.

Re-resolution occurs when one of the tables changes in such a way that the query tree stored in the *sysprocedures* table may be invalid. The datatypes, column offsets, object IDs, or other parts of the table may have changed. In this case, SQL Server must rebuild some parts of the query tree. Unfortunately, the rebuilding process results in growth of the query tree, which in turn causes the query plan to grow.

Pre-10.0 SQL Server imposes a 64-page limit on the size of both query trees and query plans. If either exceeds this limit, SQL Server issues Error 703. See “Error 703” in *SQL Server Error Messages* for more information about this error.

In releases 10.0 and later of SQL Server, there is no 64-page limit on the size of query trees and query plans. However, as query trees and query plans grow, your system could slow down considerably and

SQL Server could eventually run out of procedure cache, triggering Error 701. See “Error 701” in *SQL Server Error Messages* for more information about this error.

To curtail the growth of query trees and plans, periodically drop and re-create all stored procedures and triggers.

SQL Server re-resolves procedures after any of the following:

- **load database** is executed on the database containing the procedure.
- A table used or referenced by the procedure is dropped and re-created.
- **load database** is executed on a database in which a referenced table resides.
- A database containing a referenced table is dropped and re-created. This most commonly occurs in SQL Server releases 10.0.2 and earlier.
- A default or rule is bound or unbound to a table referred to by a query in the procedure.

6 Encyclopedia of Tasks

This chapter provides step-by-step procedures for tasks needed to recover from various disaster situations involving SYBASE system databases or the entire master device, as well as for other tasks not strictly related to disaster recovery.

Disaster Recovery Tasks

This section steps you through tasks necessary for recovery from various disaster situations involving SYBASE system databases or the entire master device.

How to Build a New Master Device

To build a new master device, execute **buildmaster**, specifying the location and size of the master device. **buildmaster** should always be run by the "sybase" user. Remember that **buildmaster** takes the size in 2K blocks (4K for Stratus). Therefore, if you want a 14MB master device, set the size parameter to 7168 2K blocks (3584 for Stratus).

◆ ***WARNING!***

Never execute buildmaster while SQL Server is running!

To build a new 14MB master device, use a command similar to one in the following table:

Platform	Command
UNIX	**buildmaster -d*device_name* -s7168**
Digital OpenVMS	**buildmaster/disk= *device_name*/size=7168**
Novell	**LOAD BLDMASTR -d*DEVICE_NAME* -s7168**
Stratus VOS	**buildmaster.pm -s3584 -d*device_name***
Novell NetWare	**load bldmastr -d *device_name* -s7168**
OS/2	**bldmastr -d *device_name* -s7168**
Windows NT	**bldmastr -d *device_name* -s7168**

If the *master* database has been altered, alter it again using exactly the same commands. The *master* database must be re-created both logically and physically to look exactly the way it did at the time of the last dump. This includes any alterations to *tempdb* or *model*.

buildmaster initializes the specified device as the SYBASE master device and creates the *master, model*, and *tempdb* databases on this device. Any information existing on the device will be overwritten.

See **buildmaster** in the *SQL Server Reference Manual*, for details.

➤ ***Note***

Be sure to execute **buildmaster** from the correct SQL Server release.

How to Rebuild *master* Database and Leave Master Device Intact

To rebuild the *master* database only and leave the master device intact, run **buildmaster** with the **-m** option (on UNIX, Stratus, or Novell) or the **/master** option (on Digital OpenVMS). Be sure to specify the correct size of the master device, not the *master* database.

The following commands build a new *master* database without changing the configuration block or initializing the master device:

Platform	Command
UNIX	**buildmaster -d*device_name* -s*device_size* -m**
Windows NT	**bldmastr -d *device_name* -s*device_size* -m**
Novell NetWare	**LOAD BLDMASTR -d*DEVICE_NAME* -s*DEVICE_SIZE* -m**
Digital OpenVMS	**buildmaster/disk=*device_name* - /master/size=*device_size***
OS/2	**bldmastr -d *device_name* -s*device_size* -m**
Stratus VOS	**buildmaster.pm -m -d*device_name***

➤ ***Note***

These commands set sort order and character set values to their defaults.

◆ ***WARNING!***

Never run the buildmaster utility while SQL Server is running.

How to Start SQL Server in Single-User Mode

To start SQL Server in single-user mode, edit a copy of the *runserver* file for the SQL Server and add the **-m** option (on UNIX) or the **masterrecover** option (on Digital OpenVMS) to the end of the **dataserver** line. On Novell, no *runserver* file is used. Instead, specify the **-m** flag on the file server command line, as shown on page 6-4.

The **startserver -m** method of starting a SQL Server in single-user mode (as documented in the *System Administration Guide*) works only on the Novell platform for SQL Server releases prior to 4.9. If the method documented in the *System Administration Guide* does not work, edit the *runserver* file.

The following example shows the *runserver* file edited to start a SQL Server named TEST in single-user mode:

On UNIX

```
#!/bin/csh -f
# Server name: TEST
# dslisten port: 4011
# master name: /dev/rsd0g
# master size: 5120
setenv DSLISTEN TEST.

exec /sybase/bin/dataserver -d/dev/rsd0g
      -e/sybase/install/errorlog_TEST -m
```

Note that the last command (starting with “exec”) must be on a single line.

On Digital OpenVMS

```
! servername: TEST
! dslisten port: 4011
define/nolog/process dslisten TEST
server:== $sybase_system:[sybase.bin]dataserver.exe
server /device=disk$database:[sybase.devices]master.dat-
      /error=sybase_system:[sybase.install]error_test.log-
      /masterrecover
```

➤ ***Note***

Create a separate *runserver* file for each SQL Server to start in single-user mode. See “How to Start SQL Server with Trace Flags” on page 6-18 for information about using *runserver* files.

Once the edited *runserver* file is created, use it to start SQL Server. For example:

Platform	Command
UNIX	**startserver -f *runserver_filename***
Digital OpenVMS	**startserver/server=*runserver_filename*/masterrecover**
Novell	**LOAD SQLSRVR -d*DEVICE_NAME* -m**

Once SQL Server is running and recovery is complete on all databases, review the error log and verify that no errors occurred.

Returning SQL Server to Multi-User Mode

To start SQL Server in multi-user mode, use the original *runserver* file without the **-m** option; on Novell, restart SQL Server without the **-m** flag.

How to Run the *installmaster/installmodel* Scripts

To execute the **installmaster** and **installmodel** scripts, located in the *$SYBASE/scripts* directory, type the command for your platform.

Platform	Command
UNIX	**isql -Usa -Psa_password < installmaster** **isql -Usa -Psa_password < installmodel**
Open VMS	**isql/u="sa"/p="sa password"/input=installmaster** **isql/u="sa"/p="sa password"/input=installmodel**
Novell NetWare	**load isql -U sa -P*sa_pswd* -S*servername* -i sys:sybase\scripts\instmstr.sql** **load isql -U sa -P*sa_pswd* -S*servername* -i sys:sybase\scripts\instmodl.sql**
OS/2	**isql -U sa -P*sa_pswd* -S*servername* -i c:\sybase\scripts\instmstr** **isql -U sa -P*sa_pswd* -S*servername* -i c:\sybase\scripts\instmodl**
Windows NT	**isql -U sa -P*sa_pswd* -S*servername* -i c:\sybase\scripts\instmstr** **isql -U sa -P*sa_pswd* -Sservername < c:\sybase\scripts\instmodl**

➤ ***Note***

On the Novell NetWare platform, each “LOAD” command must be on a single line.

The **installmaster** and **installmodel** scripts install the system procedures, set up all required SYBASE internal tables, and install the privileges for the *model* database.

How to Load the *master* Database from Backup

The SQL Server must be in single-user mode to perform this step. To load the *master* database, bring up **isql** as “sa” and execute the command:

```
1> load database master
2> from logical_dump_device_name
3> go
```

Or, if you are running System 10 or later, and the database was dumped to a remote site, see the **load database** entry in the *SQL Server Reference Manual* for information about loading the *master* database.

Once the *master* database is loaded successfully, SQL Server automatically shuts itself down and the **isql** session exits with the following message:

```
DB-Library: Unexpected EOF from SQL Server.
```

How to Restore System Table Information in *master* Database

This section is divided into two parts. The first part describes how to reestablish device and database information in the system catalog, and the second part describes how to reestablish SQL Server logins.

Restoring Device and Database Information in the System Catalog

If a **create database, alter database**, or **disk init** command has been issued since the last database dump of *master*, or if no valid dump of *master* exists, refer to the *System Administration Guide* for information on the use of the **disk reinit** and **disk refit** commands. These commands restore the system tables information contained in the *master* database, which describes all SYBASE devices and user databases.

If you kept the **disk init** scripts originally used to initialize the database devices, you can use them to formulate the **disk reinit** commands, since

disk reinit uses the same parameters. If these scripts are not available, examine the contents of *sysdevices* before a disaster and build the necessary **disk reinit** command scripts for use when needed.

➤ ***Note***

The device on which *sybsystemprocs* resides will not be included in your **disk init** script, as **sybinit** creates that device during installation. Therefore, record the values in *sysdevices* for *sybsystemprocs*, even if you plan to use your **disk init** scripts.

After the **disk reinit** command completes, compare the current contents of *sysdevices* with a copy of the *sysdevices* table that was made before the master device was lost. Since the **disk refit** command is based on the contents of that table, it is crucial that the table accurately reflect all devices.

After the **disk refit** command is complete, you must manually compare the contents of the current *sysdatabases* and *sysusages* with copies of those tables that were made prior to the loss of the master device.

Keep up-to-date copies of these tables on hand to ensure the quickest recovery after a disaster. If *sysdatabases* and *sysusages* do not match your hard-copy records, contact Sybase Technical Support for assistance.

➤ ***Note***

For System 10 or later, execute **disk reinit** on *sybsystemprocs* if it is located on a device other than master.

Reestablishing SQL Server Logins

If you have added SQL Server logins since the last database dump of *master*, or if no valid dump of *master* exists, restore the *syslogins* table:

1. Query all user databases to determine the *name* and the *suid* of each user. The **sp_addlogin** system procedure assigns an *suid* to each login in numerical order, and this *suid* is mapped to the *sysusers* table in each user database.
2. Once all *names* and *suids* are known, execute **sp_addlogin** for each user, in the appropriate order, so that newly generated logins have the same *suid* as the users in the user databases. You might have to enter dummy accounts for users whose logins have been

dropped in order to keep current users' *suid* values in the correct sequence. Drop these dummy accounts when you are done.

How to Alter the *devices* Parameter Manually

This step is necessary only if you are using a virtual device number (**vdevno**) that is greater than the default value for the **devices** configuration parameter (in this case, some of your devices will be inaccessible until you perform this step). The default value for **devices** is 10 on most platforms.

To aid in the recovery process, determine whether or not this step will be needed before an actual disaster. Do this either by examining the *device_number* column in the **sp_helpdevice** output or by executing the following query:

```
1> select max(low/power(2,24))
2> from master..sysdevices
3> go
```

If a virtual device number greater than the default is being used, you must increase the **devices** parameter before you start SQL Server. For example, if the highest **vdevno** in use is 30 and the default is 10, you must execute the following command before you start SQL Server:

Platform	Command
UNIX	**buildmaster -d*device_name* -ycnvdisks=30**
Digital OpenVMS	**buildmaster/disk=*device_name*/alter="cnvdisks=30"**
Novell	**LOAD BLDMASTR -d*DEVICE_NAME* -ycnvdisks=30**
OS/2	**bldmastr -d *device_name* -ycnvdisks=30**
Windows NT	**bldmastr -d *device_name* -ycnvdisks=30**

◆ ***WARNING!***

Never execute buildmaster while SQL Server is running.

How to Alter *tempdb*

If *tempdb* has been enlarged, but the changes are not reflected in your current *master* database, you must alter *tempdb* again to ensure that there is enough space to process your normal work load.

To help prevent errors from occurring during actual disaster recovery, record the commands you used originally to alter *tempdb*.

How to Add a SYBASE Dump Device

Dump devices are added to SQL Server via the **sp_addumpdevice** stored procedure. The syntax is:

```
1> sp_addumpdevice "device_type",
2> "dump_device_name",
3> "dump_device_location",controller_number
4> go
```

For example, use this command to add a disk dump device:

```
1> sp_addumpdevice "disk", "masterdump",
2> "/usr/sybase/master.dump",2
3> go
```

Refer to the *SQL Server Reference Manual* for more information about **sp_addumpdevice**.

Record the exact syntax of the original **sp_addumpdevice** command for each SQL Server. This helps prevent errors from occurring during disaster recovery.

How to Alter the *model* Database

Because the *model* database is created at the same time as the *master* database, no action is needed to build it. If you have made any changes to *model*, however, you must reapply them.

If you need to alter the size of the *model* database, you must alter the size of the *tempdb* database so that it is at least as big as *model*. If you attempt to start SQL Server, and *model* is bigger than *tempdb*, SQL Server will not start.

How to Execute the *reconfigure* Command

SQL Server uses three objects to store and manage its configuration parameters: the system tables, *sysconfigures* and *syscurconfigs*, and an area on the master device called the configuration block. When SQL Server starts, it reads the values from the configuration block and puts them in the *syscurconfigs* table.

The values in *sysconfigures* can be changed only by using **sp_configure**. The **reconfigure** command is used to copy values from the *config_value* column of the **sp_configure** output into the configuration block. These values take effect the next time you start SQL Server.

To illustrate how these objects are used, consider the sequence of events involving the SQL Server **memory** parameter as follows:

1. **sp_configure** shows the following row for the **memory** parameter:

   ```
   name    min    max         config value   run value
   memory  1000   2147483647  4000            4000
   ```

2. Execute the command:

   ```
   1> sp_configure memory,8000
   2> go
   ```

3. Restart SQL Server.
4. Execute the command:

   ```
   1> reconfigure
   2> go
   ```

5. Restart SQL Server.

➤ ***Note***

The restart in step 3 has no effect on any of the objects. You must use **reconfigure** to initialize the changes.

The following chart shows the state of various configuration objects at each step in the sequence:

Table 6-1: Illustration of configuration objects during sequence of events

Step	Configuration Block	*sysconfigures*	*syscurconfigs*
1) Display values with **sp_configure**	4000	4000	4000
2) Execute **sp_configure memory, 8000**	4000	8000	4000
3) Restart SQL Server	4000	8000	4000
4) Execute **reconfigure**	8000	8000	4000
5) Restart SQL Server	8000	8000	8000

How to Reset SQL Server to Its Default Configuration

To reset all of the parameters in **sp_configure** to their default values prior to starting SQL Server, use the **buildmaster** command with the

-r flag (on UNIX and Novell) or the **reconfigure** option (on Digital OpenVMS):

Platform	Command
UNIX	**buildmaster -d*device_name* -r**
Digital OpenVMS	**buildmaster/disk=*device_name*/reconfigure**
Stratus VOS	**buildmaster.pm -r -d*device_name***
Novell NetWare	**LOAD BLDMASTR -d*DEVICE_NAME* -r**
OS/2	**bldmastr -d *device_name* -rt**
Windows NT	**bldmastr -d *device_name* -r**

◆ ***WARNING!***

Never execute buildmaster while SQL Server is running.

Executing **buildmaster** while SQL Server is running causes the configuration block to be reset while leaving the rest of the master device intact.

If your SQL Server uses any devices that are mapped to a virtual device number (**vdevno**) that is greater than the default (10 on most platforms), additional steps are required to recover your server. See “How to Alter the devices Parameter Manually” on page 6-7.

Once this command has completed, you can start SQL Server. You should immediately reestablish the correct configuration values with the **sp_configure** and **reconfigure** commands, and then restart SQL Server.

How to Set SYB_BACKUP Manually in SQL Server

This procedure is needed for 10.0 and later releases of SQL Server to allow the SQL Server that is being recovered to access its Backup Server. If this step is not performed when needed, then SQL Server will not be able to process any **dump** or **load** commands.

As the SYBASE System Administrator (“sa”), execute the following commands in an **isql** session on the SQL Server that is being recovered:

```
1> use master
2> go
```

```
1> select srvname, srvnetname from sysservers
2> where srvname = "SYB_BACKUP"
3> go
```

There are three possible outcomes to this query. The following table matches each outcome to the steps you should take in that circumstance:

Outcome	Action
SQL Server returns a single row and the *srvnetname* column contains the correct reference for the Backup Server.	No further action is needed.
SQL Server returns a single row but the *srvnetname* column does **not** contain the correct reference.	Update the *srvnetname* column using the following procedure: **1> sp_configure allow,1** **2> go** **1> reconfigure with override** **2> go** **1> use master** **2> go** **1> begin transaction** **2> go** **1> update sysservers** **2> set srvnetname = <correct backup server name>** **3> where srvname = "SYB_BACKUP"** **4> go** If only one row is affected, commit the transaction. If more than one row is affected, roll back the transaction and contact Sybase Technical Support. **1> commit transaction** **2> go** **1> sp_configure allow,0** **2> go** **1> reconfigure with override** **2> go**

Outcome	Action
SQL Server returns 0 rows.	Insert the appropriate row into the *sysservers* table using the following procedure: **1> sp_configure allow,1** **2> go** **1> reconfigure with override** **2> go** **1> use master** **2> go** **1> begin transaction** **2> go** **1> insert sysservers** **2> values (number, 0, "SYB_BACKUP",** **3> backup server name)** **4> go** **1> commit transaction** **2> go** **1> sp_configure allow,0** **2> go** **1> reconfigure with override** **2> go**

Other Useful Tasks

This section steps you through tasks that are useful for resolving problems you may encounter that are not strictly related to disaster recovery.

How to Fix a Corrupted Index on System Tables

If the index on one of your system tables has been corrupted and you are running a 10.x release of SQL Server, you can use the **sp_fixindex** stored procedure to repair the index.

◆ ***WARNING!***

Before you use sp_fixindex, read each warning listed under "Read These Warnings First".

Read These Warnings First

- Do not run **sp_fixindex** in SQL Server releases earlier than 10.0, as it can **fail with major consequences**.
- Do not run **sp_fixindex** on the clustered index of the *sysobjects* table.

➤ ***Note***

You can run **sp_fixindex** on a nonclustered index on *sysobjects*, but you will encounter a known problem. For a workaround, see "Workaround for sysobjects Nonclustered Indexes" on page 6-16.

- Sybase would like to pursue the source of any persistent index corruption that is not hardware related. This debugging process requires that you do two things:
  - Leave your system catalogs untouched. Sybase must dial in to your database and examine the corruption **prior to any modifications to the system catalogs**.
  - Preserve your transaction logs. Sybase must examine your transaction logs to find the source of modifications to the pages involved.

Repairing the System Table Index

Repairing a corrupted system table index is a multi-step process; running **sp_fixindex** is one of those steps.

Here are the steps:

1. Get the object name, object ID, and index ID of the corrupted index. See “How to Find an Object Name from a Page Number” on page 6-28 for details.
2. If the corrupted index is on a system table in the *master* database, put SQL Server in single-user mode. See “How to Start SQL Server in Single-User Mode” on page 6-3 for details.
3. If the corrupted index is on a system table in a user database, put the database in single-user mode and reconfigure to allow updates to system tables.

   ```
   1> use master
   2> go

   1> sp_dboption database_name, "single user", true
   2> go

   1> sp_configure "allow updates", 1
   2> go

   1> reconfigure with override
   2> go
   ```

4. Issue the **sp_fixindex** command:

   ```
   1> use database_name
   2> go

   1> sp_fixindex database_name, object_name, index_ID
   2> go
   ```

➤ ***Note***

To run **sp_fixindex,** you must possess “sa_role” permissions.

5. Run **dbcc checktable** to verify that the corrupted index is now fixed.
6. Disallow updates to system tables:

   ```
   1> sp_configure “allow updates”, 0
   2> go
   ```

Workaround for *sysobjects* Nonclustered Indexes

Because of a known problem, running **sp_fixindex** on a nonclustered index on *sysobjects* requires several additional steps.

1. Perform steps 1–3, as described above.
2. Issue the following Transact-SQL query:

```
1> use database_name
2> go
```

```
1> select sysstat from sysobjects
2> where id = object_ID
3> go
```

3. Save the original *sysstat* value.
4. Change the *sysstat* column to the value required by **sp_fixindex**:

```
1> update sysobjects
2> set sysstat = sysstat | 4096
3> where id = object_ID
4> go
```

5. Run **sp_fixindex**:

```
1> sp_fixindex database_name, object_name, index_ID
2> go
```

6. Restore the original *sysstat* value:

```
1> update sysobjects
2> set sysstat = sysstat_ORIGINAL
3> where id = object_ID
4> go
```

7. Run **dbcc checktable** to verify that the corrupted index is now fixed.
8. Disallow updates to system tables:

```
1> sp_configure "allow updates", 0
2> go
```

How to Rescue Data from a Corrupted Table

This section describes the steps needed to copy data from a corrupted table into a new table or file. Note that you will probably be able to copy only **some** of your data.

Back Up Data to a New Table

Copy the data from the corrupted table into a new table by creating a dummy table, copying the old data into the dummy table, and then deleting the old table.

You can create the new table in any database (except *model*) where enough space is available. Follow these steps:

1. Check the table size that you want to copy:

   ```
   1> sp_spaceused table_name
   2> go
   ```

2. Check the amount of available space in the database in which you plan to create the new table:

   ```
   1> use database_name
   2> go
   ```

   ```
   1> sp_spaceused
   2> go
   ```

   The easiest way to copy the table into a new one is to select all the data from your corrupted table into a temporary table. This way, you can skip step 3.

   If space is too limited to create your table in any database, you may **bcp** (copy out) the data from the table, drop the table, recreate it, and **bcp** (copy in) the data into it. See **bcp** in the *SYBASE SQL Server Utility Programs* manual for your platform.

3. Enable the **select into/bulkcopy** option on the database where you want to create the new table. You do not need to enable the **select into/bulkcopy** option on *tempdb*, as *tempdb* already has this option enabled. For more information about enabling the **select into/bulkcopy** option on a database, refer to "Error 268" in *SQL Server Error Messages*.

   After you have run a **select into** command or used **bulkcopy** to move data into a database, you cannot perform a regular transaction log dump. Also, once you have made unlogged changes to your database, you must perform a **dump database**, since changes would not be recoverable from transaction logs.

   Just setting the **select into/bulkcopy** option to "on" does not block the use of a regular **dump transaction**. You can still use **dump transaction... with truncate_only.**

◆ ***WARNING!***

Be careful about running select into across databases if you have column names that exist in both databases, as this may cause problems.

4. Copy the old table into the new table:

```
1> select * into database_name..new_table
2> from old_table
3> go
```

Or, if you select all the data into a temporary table:

```
1> select * into tempdb..new_table from old_table
2> go
```

Back Up Data to an Operating System File

To back up data into an operating system file, perform the following steps:

1. Use **bcp** to copy the data from the table into a file. For information about **bcp**, see **bcp** in the *SQL Server Reference Manual*.
2. Use **sp_rename** to rename the old table.
3. Use **bcp** to copy the file into a new table, using the old table name.
4. Drop the renamed table.

How to Start SQL Server with Trace Flags

Follow the instructions in this section to start SQL Server with a trace flag. If you have a UNIX or Digital OpenVMS system, you can modify the runserver file to start SQL Server with a trace flag. OS/2 and Novell systems use the command line to start SQL Server with a trace flag.

Look for the section below that matches your operating system.

◆ ***WARNING!***

Start SQL Server with a trace flag only when instructed to do so in this manual or as directed by Sybase Technical Support or an EBF letter. Using these flags at any other time may create serious problems.

Modifying the Runserver File for UNIX

1. Make a copy of the runserver file. A common naming convention for this new file is *RUN_SERVERNAME_TRACEFLAG*. For example, if you wanted to start a SQL Server named PRODUCTION with the 1603 trace flag (which disables asynchronous I/O), you could copy your existing runserver file into a file named *RUN_PRODUCTION_1603*.
2. Edit the new runserver file to include the desired trace flag.

   The sample modified runserver file below includes the 1603 trace flag for a SQL Server named PRODUCTION (substitute the correct values for your installation, including the correct trace flag number):

   ```
       #! /bin/csh -f
       # Server name:  PRODUCTION
       # dslisten port:  4011
       # master name:  /dev/rid001d
       # master size:  5120

       setenv DSLISTEN PRODUCTION

   $SYBASE/bin/dataserver -d/dev/rid001d
   -e$SYBASE/install/errorlog_PRODUCTION -T1603
   ```

   The last element of the last line activates the trace flag, which is flag 1603 in this example.
3. Use the **startserver** command to start SQL Server with the modified runserver file:

   ```
   % startserver -f RUN_PRODUCTION_1603
   ```

➤ ***Note***

The **startserver** command must be on one line.

4. After you have completed corrections, restart SQL Server with your normal runserver file.

Modifying the Runserver File for Digital OpenVMS

1. Make a backup copy of your runserver file, and then edit the copy of the file to include the desired trace flag. The sample modified runserver file below includes the 1603 trace flag (which

disables asynchronous I/O) for a SQL Server named PRODUCTION:

```
$ define/nolog sybase_system PRODUCTION
$ @sybase_system:[sybase.install]sylogicals.com
$ ! Server name:   PRODUCTION
$ ! Dslisten port:   PRODUCTION
$ define/nolog/process DSLISTEN PRODUCTION
$server :== $sybase_system:[sybase.bin]dataserver.exe
$server /device=disk$database:[sybase.devices]master.dat -
    /error=sybase_system:[sybase.install]error_production.log -
    /trace=1603
```

The last element of the last line activates the trace flag, which is flag 1603 in this example.

2. Use the **startserver** command to start SQL Server with the modified runserver file:

```
VMS$ startserver /server=production
```

3. Change the name of the modified runserver file to store it as a backup should you need to run SQL Server with this trace flag again. Then restore the backup copy of the original runserver file you made in step 1 to its original name.
4. After you have completed corrections, restart SQL Server with your normal runserver file.

Using Trace Flags in Novell

To start SQL Server with a special trace flag, add the trace flag to the **load** command on the console command line. For example:

```
:load SQLSRVR -dDEVICE_NAME -Ttrace_flag_number
```

Substitute your site’s master device physical device name and the trace flag number you want to use.

Using Trace Flags in OS/2

To start SQL Server with trace flags in OS/2, use a command similar to the following:

```
sqlserver /d device_name /Ttrace_flag_number
```

Substitute your site’s master device physical device name and the trace flag number you want to use.

How to Reload a Suspect Database

If all other methods of restoring a user database marked “suspect” have failed, use this section to reload the suspect database from a known, clean backup.

SQL Server Release 4.8 or Later

Reload the suspect user database from backup by following the steps in the section on database recovery in the *System Administration Guide*. It is very important to follow that procedure to ensure that the segment sizes and locations are created in the proper order, or your database will not reload properly.

If you cannot drop the database using the normal procedure, use the **dbcc dbrepair** command. See “How to Drop a Database When drop database Fails” on page 6-21.

For more information about reloading databases, see “Error 2558” in *SQL Server Error Messages*.

SQL Server Release 4.2 or Earlier

When SQL Server marks a database suspect, that database is not accessible until the suspect status is corrected. Contact Sybase Technical Support for assistance.

How to Drop a Database When *drop database* Fails

Follow the steps below to drop a database when **drop database** fails to drop it. Do not use the procedures in this section unless directed to do so by this *Troubleshooting Guide*, or unless there is no critical data in the database.

1. Log in as the “sa”.
2. Check to make sure the database has been marked “suspect.” The following query produces a list of all databases which are marked suspect:

```
1> select name from master..sysdatabases
2> where status & 256 = 256
3> go
```

3. If the database is marked “suspect”, go to step 4. If it is not marked “suspect”, mark it in one of the following ways:

1. Execute the **sp_marksuspect** stored procedure discussed under "How to Mark a Database "suspect"" on page 6-39, and restart SQL Server to initialize the change.

2. Use the procedure below:

```
1> sp_configure "allow updates", 1
2> go

1> reconfigure with override
2> go

1> use master
2> go

1> begin tran
2> update sysdatabases set status = 256
3> where name = "database_name"
4> go
```

Verify that only one row was affected and commit the transaction:

```
1> commit tran
2> go
```

Reset the **allow updates** option of **sp_configure**:

```
1> sp_configure "allow updates", 0
2> go

1> reconfigure with override
2> go
```

Restart SQL Server to initialize the change.

4. Remove the database:

```
1> dbcc dbrepair(database_name,dropdb)
2> go
```

dbcc dbrepair sometimes displays an error message even though it successfully drops the database. If an error message occurs, verify that the database is gone by executing the **use database_name** command. This command should fail with a 911 error, since you dropped the database. If you find any other error, contact Sybase Technical Support.

How to Fix and Prevent Allocation Errors

This section describes allocation errors, how to fix them, and how to prevent them from reoccurring.

Understanding Allocation Errors

The **dbcc checkalloc**, **dbcc tablealloc**, and **dbcc indexalloc** commands check the consistency of the allocation structures in a database. If an inconsistency between information in the page chain of an object and information in the allocation structures of that object is detected, an error is displayed. Additionally, if **dbcc checkalloc** is run while the database is not in single-user mode, errors that do not really exist (spurious errors) may be reported. Spurious errors may be reported when changes in the database occur while **dbcc checkalloc** is running.

Allocation errors 2521, 2540, 2546, 7939, 7940, and 7949 have different levels of severity, but they should all be corrected.

Fixing Allocation Errors

➤ ***Note***

dbcc checkalloc with the **fix** option was **dbcc fix_al** in SQL Server releases previous to 10.0. **dbcc fix_al** is supported only in SQL Server 4.9.1. It has been used successfully in release 4.8; however, we cannot guarantee the safety of **dbcc fix_al** in 4.8.

Follow these steps to correct any allocation error that has occurred, including 2521, 2540, 2546, 7939, 7940, and 7949:

1. Set the database that encountered the error in single-user mode. If the error was on the *master* database, set it to single-user mode by shutting down and restarting SQL Server in single-user mode. See "How to Start SQL Server in Single-User Mode" on page 6-3 for instructions. If the database is a user database, use this procedure:

```
1> sp_dboption database_name, single, true
2> go

1> use database_name
2> go

1> checkpoint
2> go
```

➤ ***Note***

dbcc checkalloc with the **fix** option fails with Error 2595 if the database is not set in single-user mode. If you cannot run SQL Server with the database in single-user mode, refer to *SQL Server Error Messages* for the particular error you are trying to correct, or call Sybase Technical Support.

2. Run **dbcc checkalloc** with the **fix** option to correct the error:

```
1> use master
2> go
```

```
1> dbcc checkalloc(database_name, fix)
2> go
```

3. Reset the database from single-user mode. To reset the *master* database, shut down and restart SQL Server without the special single-user mode procedure. To reset a user database, use the following procedure:

```
1> sp_dboption database_name, single, false
2> go
```

```
1> use database_name
2> go
```

```
1> checkpoint
2> go
```

➤ ***Note***

For large databases, you may want to execute the commands in steps 1–3 from a script file, which allows you to save the results for future reference.

4. Examine the **dbcc checkalloc** output. If there are any errors, refer to *SQL Server Error Messages*, or contact Sybase Technical Support.

Preventing Allocation Errors

This section provides two strategies for preventing allocation errors 2521, 2540, 2546, 7939, 7940, and 7949:

- See “Single-User Mode Method (Spurious and Non-spurious Errors)” on page 6-25 if the database can be placed in single-user mode to perform maintenance tasks.
- See “Multi-User Mode Method (Spurious Errors Only)” on page 6-26 if you cannot invoke single-user mode on the database in question (for example a 24-hour production site).

Without single-user mode, you cannot prevent non-spurious error messages from occurring.

Single-User Mode Method (Spurious and Non-spurious Errors)

If you can run **dbcc checkalloc** in single-user mode, replace each occurrence of **dbcc checkalloc** in scripts and procedures with **dbcc checkalloc** with the **fix** option, as follows:

```
1> use master
2> go
1> sp_dboption database_name, single, true
2> go
```

➤ ***Note***

Use **dbcc checkalloc** with the **fix** option while in a database other than the one that is being repaired.

```
1> use database_name
2> go
1> checkpoint
2> go
1> use master
2> go
1> dbcc checkalloc(database_name, fix)
2> go
1> sp_dboption database_name, single, false
2> go
1> use database_name
2> go
1> checkpoint
2> go
```

Before you implement this strategy, consider these facts:

- **dbcc checkalloc** with the **fix** option must be run in single-user mode.
- Because **dbcc checkalloc** with the **fix** option may report other errors, Sybase recommends that you save the output from the **dbcc checkalloc** command and examine it.
- **dbcc checkalloc** with the **fix** option is the same program as **dbcc checkalloc**, except that **dbcc checkalloc** with the **fix** option requires single-user mode and fixes errors instead of just reporting them. **dbcc checkalloc** with the **fix** option is not slower than **dbcc checkalloc**.

- Because the *master* database is usually updated less frequently, allocation errors occur much less often. Therefore, you may not need to use this strategy on *master*. If you do use it on *master*, see "How to Start SQL Server in Single-User Mode" on page 6-3 of this guide for instructions on how to activate single-user mode (it cannot be invoked via **sp_dboption** on *master*).
- You do not ever need to run **dbcc checkalloc** after **dbcc checkalloc** with the **fix** option to ensure that the errors were corrected.
- Although no actual users are logged on, you may not be able to enable single-user mode if there are processes still active.

If you have databases on which you cannot run allocation checks in single-user mode, use the following procedure to eliminate the spurious allocation errors that can occur when **dbcc checkalloc** is run in multi-user mode.

Multi-User Mode Method (Spurious Errors Only)

If your site does not allow single-user operation (such as a 24-hour production SQL Server), you cannot completely prevent allocation errors, but you can prevent spurious errors. Use the following strategy to stop occurrences of spurious allocation errors:

1. You can use this strategy only in SQL Server 4.9.1 at the following EBF Rollup level or greater:

Platform	EBF Rollup Level
SunOS 4.x (Sun Solaris 1.x)	1440
HP 9000 Series 800	1441
RS6000	1442
Digital OpenVMS	1444
AT&T SVR4	1597

   If your system is not running at the correct level for your platform, contact Sybase Technical Support to obtain the appropriate EBF rollup.

2. Replace each occurrence of **dbcc checkalloc** in scripts and procedures with the following:

```
1> dbcc traceon (2512)
2> go
```

```
1> dbcc checkalloc ( database_name )
2> go
```

```
1> dbcc traceoff (2512)
2> go
```

```
1> use database_name
2> go
```

```
1> dbcc tablealloc (syslogs)
2> go
```

This procedure prevents **dbcc checkalloc** from examining the *syslogs* table, where the spurious errors originate (**dbcc tablealloc** checks *syslogs* instead). If you get genuine allocation errors, refer to *SQL Server Error Messages* for instructions.

Before you implement this strategy, consider these facts:

- This strategy is unnecessary if you can run the database in single-user mode. If you can run the database in single-user mode, use the strategy described in "Single-User Mode Method (Spurious and Non-spurious Errors)" on page 6-25.
- Because the *master* database is usually updated less frequently than user databases, allocation errors occur much less frequently. Therefore, this strategy may be unnecessary on *master*.
- Check the status of all the databases at your site. You may be able to do the allocation checks in single-user mode and use the simpler single-user mode strategy described in "Single-User Mode Method (Spurious and Non-spurious Errors)" on page 6-25.

Syntax for *dbcc checkalloc* with the *fix* Option

This section explains only **dbcc checkalloc** with the **fix** option. See the *SQL Server Reference Manual* for information about **dbcc** and all its other keywords and options.

Function

dbcc checkalloc with the **fix** option detects and fixes allocation errors in databases. It is supported only in SQL Server release 4.9.1 or later, although it has been used successfully at some sites in release 4.8.1.1.

Syntax

```
dbcc checkalloc(database_name, fix)
```

Example

```
1> sp_dboption database_name, single, true
2> go

1> use database database_name
2> go

1> checkpoint
2> go

1> dbcc checkalloc(database_name, fix)
2> go

1> use master
2> go

1> sp_dboption database_name, single, false
2> go

1> checkpoint
2> go
```

Comments

Databases must be in single-user mode or **dbcc checkalloc** with the **fix** option will fail with error 2595.

How to Find an Object Name from a Page Number

Some SQL Server error messages only specify a logical page number and do not indicate the table or index name to which the page belongs. This section describes how to determine to which object a particular database page belongs.

You can use **dbcc checkalloc** and **dbcc checkdb** to identify the table name and index IDs. A quicker method is described below. However, this method only tells you the table or index associated with a particular page number.

Suppose you encounter this error message:

```
Error 644, Severity 21, State 1 The non_clustered
leaf row entry for page 1342 row 6 was not found
in index page 944 indexid 3 database 'production'
```

The error message implies that a nonclustered index is corrupt. However, the corresponding table name or index name is not given; only a page number (944) and an index ID (3) are given.

To determine which table and index is involved, follow these steps:

1. Log into SQL Server as “sa.”
2. Enable trace flag 3604 to allow **dbcc** output to appear at your terminal:

   ```
   1> dbcc traceon(3604)
   2> go
   ```

3. Use **dbcc page** to display information about the page in question.

   The syntax of this command is:

   ```
      dbcc page (database_name, page_number)
   ```

➤ ***Note***

The **dbcc page** command is not a supported feature and Sybase Technical Support cannot answer any questions regarding any values other than object ID and index ID.

For example, to find information about page 944 (the index page indicated in the error message) in the *production* database, execute the following commands:

```
1> dbcc page(production, 944)
2> go
```

```
PAGE: Page not found in cache - read from disk.
BUFFER: Buffer header for buffer 0x2c4b30 page=0x540800 bdnew=0x0
bold=0x0 bhash=0x0 bnew=0x0 bold=0x0 bvirtpg=7092 bdbid=6
bpinproc=0 bkeep=0 bspid=0
bstat=0x0000 bpageno=0

PAGE HEADER: Page header for page 0x540800
pageno=944 nextpg=0 prevpg=0 objid=9051068 timestamp=0001
000c70f2
nextrno=32 level=0 indid=3 freeoff=52 minlen=7
page status bits: 0x2,

DBCC execution completed. If DBCC printed error messages, see
your System Administrator.
```

◆ ***WARNING!***

Be sure to provide the correct page number, as an incorrect page number could cause SQL Server to stack trace.

4. Translate the object ID (*objid*) into a table name:

```
1> use database_name
2> go

1> select object_name(9051068)
2> go

-------------
bad_table
```

5. Translate the index ID (*indid*) into an index name, if applicable:

```
1> use production
2> go

1> select name
2> from sysindexes
3> where id = objid
4> and indid = indid
5> go
```

6. Refer to the table below to determine the index type:

Index ID	Meaning
0	Actual table data
1	Clustered index
2- 250	Nonclustered indexes
255	Text/image page

Index ID 3 from the example corresponds to a nonclustered index. If the index ID is 0, the page does not belong to an index.

7. Disable trace flag 3604:

```
1> dbcc traceoff(3604)
2> go
```

How to Interpret *sp_who* Output

The following table shows the sleep classifications that appear in **sp_who** output:

Table 6-2: Sleep classifications from sp_who output

Classification	Meaning
send sleep	SQL Server process is going to sleep until the send is completed by the network service task.

Table 6-2: Sleep classifications from sp_who output

Classification	Meaning
recv sleep	SQL Server process is sleeping until it receives something from the client (this is the most common status).
lock sleep	SQL Server process is waiting for locks (resource, logical, semaphore, and so on) to be released.
alarm sleep	SQL Server process is waiting for an alarm to wake it up (user executed a **waitfor delay** command).
sleeping	SQL Server process is waiting for a resource to post network or disk I/O.

Device Administration Issues

This section discusses issues to consider when choosing between raw partitions and UNIX files and describes how to use partitions correctly.

How to Choose Between Raw Partitions and UNIX Files

A raw partition on a UNIX system is a part of the disk where there is no file system. Although SQL Server can use UNIX files for database devices, Sybase strongly recommends using raw partitions instead.

Most UNIX systems use a buffer cache for disk I/O. Writes to disk are stored in the buffer and may not be written to disk immediately. If SQL Server completes a transaction and sends the result to a UNIX file, the transaction is considered complete even though the UNIX buffer cache may not have been written to disk. If the system crashes before this buffer cache is written, you lose data. In this situation, SQL Server has no way of knowing that the write to disk eventually failed, and the transaction is not rolled back. In addition, some UNIX operating systems do partial writes. In that case, if the system crashes, the SYBASE device will be corrupted.

Using raw partitions for SYBASE devices allows SQL Server to process its own I/O requests, without having to go through the UNIX buffering scheme. In this way, SQL Server knows exactly what portions of a transaction completed or failed in the event of a system crash. If SYBASE devices use UNIX files, disaster recovery by SQL Server is not guaranteed.

See the *System Administration Guide Supplement* or your operating system documentation for more information.

Correct Use of Raw Partitions

If you choose to use raw partitions, examine your operating system's use of partitions carefully. Otherwise, you may overwrite valuable data. In particular, avoid the following situations:

- Partition is already in use
- Partition overlaps with another partition
- Operating system is using partition for swap space
- A file system is mounted on the partition

Partition Is Already In Use

Ask your UNIX system administrator what the partition was originally configured for and make sure that it was not designated to serve for any other purpose except for the use of your SQL Server. If your partition is used for any other purpose, most of the information it stores might be corrupted or destroyed.

Partition Overlaps with Another Partition

Verify that the partition you intend to use does not share cylinders with another partition. In particular, watch for the following scenarios:

- On some UNIX systems (for example, SunOS BSD), partition *c* is, by convention, defined to be the whole disk, so it is expected that partition *c* will overlap all the other partitions.

  If you are using partition *c* for your database device, do not use any other partitions on that drive, or check with your UNIX System Administrator to make sure that partition *c* is not defined as being the whole disk.
- On other UNIX systems (AT&T SVR4), partition *s6* is defined to be the whole disk.

Operating System Is Using Partition for Swap Space

Refer to your operating system administration guide for steps to determine whether a partition is being used for swap space.

For example, on AT&T SVR4 and HP, check to see if your partition is included in the output that is generated, using the following commands:

On AT&T SVR4:

```
% /etc/swap -l
```

On HP:

```
% /etc/swapinfo or /usr/sam/bin/swapinfo
```

These commands report information on swap partitions only if the entries are found in the file system table.

If the output of these commands includes the device name associated with your database partition, then the device is being used for swap space. Ask your operating system administrator which partition you may use for your database. For more information on how to choose raw partitions, see the *SYBASE SQL Server Installation Guide* for your platform.

A File System Is Mounted on the Partition

Check to see if your partition is included in the output generated by the following command(s):

```
% df
% /etc/mount
```

If the output from these commands includes the device name associated with your database partition, ask your operating system administrator to unmount the file system from the partition **or** to help you choose another disk partition. Note that using the partition as a raw database partition will destroy all file system information that was there.

Getting Information About Your Partition

There are several ways to determine how a raw partition is being used:

- Interview your operating system administrator
- Examine your file system table
- Examine the partition map

Examine the File System Table

The file system table name varies by platform, as described in the following table:

Platform(s)	File System Table Name
SunOS 4.x (BSD) DG UX	*/etc/fstab*
Sun Solaris 2.x (SPARC) AT&T SVR4	*/etc/vfstab*
AIX	*/etc/filesystems*
HP	*/etc/checklist*

➤ ***Note***

Good commenting in the file system table helps prevent most disk partition errors.

Examine the Partition Map

Each partition includes a partition map, which is usually in the first sector of the first cylinder.

The partition includes the partition map, which is usually in the first sector of the first cylinder. Refer to your operating system administration guide for steps to determine at what cylinder a partition starts.

The following table lists the commands used on selected operating systems to check where a partition starts:

Platform(s)	Command Syntax
SunOS 4.x (BSD)	**% /etc/dkinfo rsd0d** where *rsd0d* is the device name associated with the database partition
Sun Solaris 2.x (SPARC) AT&T SVR4	**% /usr/sbin/prtvtoc /dev/rdsk/c0t1d0s0**

➤ ***Note***

If you are running the Logical Volume Manager (LVM) on an AIX operating system, verify that the first AIX cylinder of your raw partition is free (except for the master device) and is available for the use of the LVM configuration. In order to do this, make sure that *vstart* = 2 (one AIX cylinder = 2 Sybase pages) for all user-defined disks.

See your operating system documentation for more information about disk partition administration. The *SYBASE SQL Server Installation Guide* contains additional information about choosing a raw partition for your database device.

Other Situations to Avoid

Do not let multiple SQL Server devices or mirrors use the same partition. Make a list of all partitions used by all SQL Servers on the machine and look for duplicates. The *$SYBASE/install/RUN_SERVERNAME* (or equivalent) file contains the master device name. Use the stored procedure **sp_helpdevice** in each SQL Server to find all the database devices and mirrors in use by that SQL Server.

Having two or more SQL Servers on the same machine with two or more SYBASE System Administrators increases the likelihood of this problem. Any process logged in as “sybase” can write to that partition since the user “sybase” owns it. To minimize the risk, keep a log of all the partitions in use by UNIX and by SQL Server. Establish procedures for updating the log when any configuration changes are made.

As an extra security check, make sure that the permissions for the device are read and write **only** by the “sybase” user. Then, if another user attempts to write anything to that partition, no damage will occur.

How to Move Databases and Segments With Disk Mirroring

Although the primary purpose of disk mirroring is to expedite recovery, it has other uses:

- To move a non-mirrored database
- To move a log segment to another device

Moving a Non-Mirrored Database

The usual procedure to move a database on a non-mirrored device is as follows:

1. Dump the database
2. Drop the database
3. Drop the device
4. Initialize the new devices
5. Create the database on the new devices
6. Load the database

You can do this more easily by using disk mirroring. For example, to move a database from *disk1* and *disk2* to *disk3* and *disk4*, use the following commands:

```
1> disk mirror name = "disk1",
2> mirror = "/usr/u/sybase/disk3"
3> go

1> disk mirror name = "disk2",
2> mirror = "/usr/u/sybase/disk4"
3> go

1> disk unmirror name = "disk1",
2> side = primary, mode = remove
3> go

1> disk unmirror name = "disk2",
2> side = primary, mode = remove
3> go
```

This technique is particularly helpful when moving the log device for a database to another device, as explained in the section that follows.

➤ ***Note***

This procedure is useful, provided that the database in question uses devices *disk1* and *disk2* only. Any other databases stored on devices *disk1* and *disk2* will also be moved.

Moving a Log Segment to Another Device

The *System Administration Guide* explains how to move the log segment of a database to another device by altering the database to include the new device and changing the mapping in the system tables to force future allocations for log pages to go to the new device. The old device is still in use by the database, but only for data objects, not for logging.

Sometimes you must completely remove a device used by a particular database. You can do this by rebuilding the database, but removing a disk is sometimes easier to do using disk mirroring.

For example, if a log segment is on a logical device named *logdisk* and you want to completely remove the physical device it is mapped to, and replace it with a new physical disk (in this example */usr/u/sybase/newdisk*), use the following commands:

```
1> disk mirror name = 'logdisk',
2> mirror = '/usr/u/sybase/newdisk'
3> go
```

```
1> disk unmirror name = 'logdisk',
2> side = primary, mode = remove
3> go
```

The commands above remap the logical device *logdisk* from the original physical device to the new physical device—in this example, */usr/u/sybase/newdisk*.

For more information on disk mirroring, see Chapter 10, “Disk Mirroring.”

How to Gather Information About Read/Write Errors

The commands to create a procedure called **sp_diskblock**, which translates a SYBASE virtual disk and block number into the corresponding SYBASE device, database, and logical page number, are shown on page 6-38. Use **sp_diskblock** to gather information about read or write errors that SQL Server might encounter. See “Read / Write Error” in *SQL Server Error Messages* for more information.

sp_diskblock collects information from the system tables of the SQL Server on which it is executed; therefore, you must execute it on the SQL Server that has the read / write error.

◆ ***WARNING!***

The sp_diskblock stored procedure is provided for your information—it is not supported at this time.

Before You Create and Execute *sp_diskblock*

Before creating and executing **sp_diskblock**, note the following:

- For pre-10.0 SQL Server, create **sp_diskblock** in the *master* database. For 10.0 or later SQL Server, create **sp_diskblock** in the *sybsystemprocs* database.
- If you change the name of the procedure, make sure the new procedure name begins with "sp_".
- Review the *Transact-SQL User's Guide* explanation of how to create and execute stored procedures.

Syntax

sp_diskblock *virtual_disk*, *block_number*

Sample

```
1> sp_diskblock 4, 871
2> go
```

```
Virtual disk 4, block 871 corresponds to:
Logical page 1895 in the "production" database
(dbid=4) on device "main".
```

Stored Procedure Code

```
CREATE PROC sp_diskblock @disk int, @block int AS
DECLARE @low    int,
        @dname  varchar(30),
        @msg    varchar(90),
        @lpage  int,
        @dbid   int,
        @segmap int
SELECT  @low = low,  @dname = name
FROM master.dbo.sysdevices WHERE low/16777216 = @disk
     and cntrltype = 0

IF ( @low IS NULL )
BEGIN
     SELECT @msg = 'Virtual device ' + CONVERT(varchar, @disk)
```

```
      + ' does not exist on this server.'
     PRINT @msg
     RETURN (1)
END
ELSE
BEGIN
     SELECT  @lpage = lstart + @block + @low - vstart,
          @dbid = dbid, @segmap = segmap
          FROM master.dbo.sysusages WHERE(@block + @low)>= vstart
          AND (@block + @low) <= (vstart + size)
     IF ( @dbid IS NULL )
          BEGIN
           SELECT @msg = 'Block ' + CONVERT(varchar, @block)
                   +' on disk "' + @dname
                   + '" is currently notin use for any database.'
           PRINT @msg
           RETURN (1)
          END
     ELSE
          BEGIN
           SELECT @msg = "Virtual disk" + convert(varchar,@disk)
                   + ", block "  + convert(varchar,@block)
                   + " corresponds to:"
           PRINT @msg
           SELECT @msg ='Logical page ' + convert(varchar,@lpage)
                   + ' in the "'  + DB_NAME(@dbid)
                   + '" database (dbid=' + convert(varchar(3),@dbid)
                   + ') on device "' + @dname + '".'
            PRINT @msg
          END
END
RETURN (0)
```

How to Mark a Database “suspect”

The commands to create a procedure called **sp_marksuspect**, which turns on the suspect status bit on the specified database, are described on page 6-41.

Use **sp_marksuspect** to prepare a damaged database that is to be dropped with **dbcc dbrepair.**

◆ ***WARNING!***

The sp_marksuspect stored procedure is provided for your information—it is not supported at this time.

Before You Create and Execute *sp_marksuspect*

Before creating and executing **sp_marksuspect**, note the following:

- For pre-10.0 SQL Servers, create **sp_marksuspect** in the *master* database. For 10.0 or later SQL Servers, created **sp_marksuspect** in the *sybsystemprocs* database.
- Since this procedure modifies the system catalog, you must enable updates to the catalog before executing the procedure. Use the procedure below to enable updates:

  ```
  1> use master
  2> go
  
  1> sp_configure "allow updates", 1
  2> go
  
  1> reconfigure with override
  2> go
  ```
- If you change the name of the procedure, make sure the new procedure name begins with “sp_”.
- Review the *Transact-SQL User’s Guide* explanation of how to create and execute stored procedures.

After You Execute *sp_marksuspect*

Once the procedure is created successfully, updates to the system catalog should be immediately disabled as follows:

```
1> sp_configure "allow updates", 0
2> go

1> reconfigure
2> go
```

Syntax

sp_marksuspect *database_name*

Example

```
1> sp_marksuspect PRODUCTION
2> go

Database 'PRODUCTION' has been marked suspect!
WARNING: This database should now be dropped
 via dbcc dbrepair!
```

Stored Procedure Code

```
CREATE PROC sp_marksuspect @dbname varchar(30) AS
   DECLARE @msg varchar(80)
   IF @@trancount > 0
     BEGIN
      PRINT "Can't run sp_resetstatus from within a transaction."
      RETURN (1)
     END

   IF suser_id() != 1
     BEGIN
      SELECT @msg = "You must be the System Administrator (SA)
      SELECT @msg = @msg + "to execute this procedure."
      PRINT @msg
      RETURN (1)
     END

   IF (SELECT COUNT(*) FROM master..sysdatabases
      WHERE name = @dbname) != 1
     BEGIN
      SELECT @msg = "Database '" + @dbname + "' does not exist!"
      PRINT @msg
      RETURN (1)
     END

IF (SELECT COUNT(*) FROM master..sysdatabases
      WHERE name = @dbname and status & 256 = 256) = 1
     BEGIN
      SELECT @msg = "Database '" + @dbname + "' "
      SELECT @msg = @msg + "is already marked suspect."
      PRINT @msg
      RETURN (1)
     END
   BEGIN TRAN
     update master..sysdatabases set status = status|256
       WHERE name = @dbname
   IF @@error != 0 or @@rowcount != 1
     ROLLBACK TRAN
   ELSE
    BEGIN
    COMMIT TRAN
    SELECT @msg = "Database '" + @dbname + "' has been marked suspect!"
    PRINT @msg
    PRINT " "
    SELECT @msg = "WARNING: This database should now be dropped"
    SELECT @msg = @msg + "via dbcc dbrepair."
    PRINT @msg
    PRINT " "
   END
```

How to Reset a Database's "suspect" Status

The commands to create a procedure called **sp_resetstatus**, which turns off the "suspect" flag on a database while leaving all other database options intact, are shown on page 6-43.

Use **sp_resetstatus** only when so instructed in this manual or by Sybase Technical Support. Otherwise, you may damage your database.

◆ ***WARNING!***

The sp_resetstatus stored procedure is provided for your information—it is not supported at this time.

Before You Create and Execute *sp_resetstatus*

Before creating and executing **sp_resetstatus**, note the following:

- For pre-10.0 SQL Servers, create **sp_resetstatus** in the *master* database. For 10.0 or later SQL Servers, created **sp_resetstatus** in the *sybsystemprocs* database.
- You must have **sa_role** to execute this procedure.
- Since this procedure modifies the system catalog, you must enable updates to the catalog before executing the procedure. Use the procedure below to enable updates:

```
1> use master
2> go

1> sp_configure "allow updates", 1
2> go

1> reconfigure with override
2> go
```

- If you change the name of the procedure, make sure the new procedure name begins with "sp_".
- Review the *Transact-SQL User's Guide* explanation of how to create and execute stored procedures.

After You Execute *sp_resetstatus*

After successfully executing this procedure, you must do two things:

1. Immediately shut down SQL Server.
2. Restart SQL Server and immediately disable updates to the system catalog as follows:

   ```
   1> sp_configure "allow updates", 0
   2> go
   ```

   ```
   1> reconfigure
   2> go
   ```

Syntax

sp_resetstatus ***database_name***

Example

```
1> sp_resetstatus PRODUCTION
2> go
```

```
Database 'PRODUCTION' status reset!
WARNING: You must reboot SQL Server prior to
         accessing this database!
```

Stored Procedure Code

```
CREATE PROC sp_resetstatus @dbname varchar(30) AS
DECLARE @msg varchar(80)

IF @@trancount > 0
    BEGIN

      PRINT "Can't run sp_resetstatus from within a transaction."
      RETURN (1)
    END

IF suser_id() != 1
    BEGIN
      SELECT @msg =  "You must be the System Administrator (SA)"
      SELECT @msg = @msg + " to execute this procedure."
      PRINT @msg
      RETURN (1)
    END

IF (SELECT COUNT(*) FROM master..sysdatabases
      WHERE name = @dbname) != 1
    BEGIN
      SELECT @msg = "Database '" + @dbname + "' does not exist!"
      PRINT @msg
      RETURN (1)
    END
```

```
IF (SELECT COUNT(*) FROM master..sysdatabases
      WHERE name = @dbname AND status & 256 = 256) != 1
    BEGIN
      PRINT "sp_resetstatus may only be run on suspect databases."
      RETURN (1)
    END

BEGIN TRAN
    UPDATE master..sysdatabases SET status = status^256
     WHERE name = @dbname
    IF @@error != 0 OR @@rowcount != 1
     ROLLBACK TRAN
    ELSE

       BEGIN
         COMMIT TRAN
         SELECT @msg = "Database '" + @dbname + "' status reset!"
         PRINT @msg
         PRINT " "
         PRINT "WARNING: You must reboot SQL Server prior to  "
         PRINT "          accessing this database!"
         PRINT " "
       END
```

How to Find a Device's Virtual Device Number

The commands to create a procedure called **sp_vdevno**, which finds the virtual device number of a given device, are shown on page 6-45.

sp_vdevno returns results similar to the following:

```
1> sp_vdevno
2> go

vdevno       name          status
----------   -----------    ------
        0    master            2
        4    user_disk4        3
```

◆ ***WARNING!***

The sp_vdevno stored procedure is provided for your information—it is not supported at this time.

Before You Create and Execute *sp_vdevno*

Before creating and executing **sp_vdevno**, note the following:

- For pre-10.0 SQL Servers, create **sp_vdevno** in the *master* database. For 10.0 or later SQL Servers, created **sp_vdevno** in the *sybsystemprocs* database.
- If you change the name of the procedure, make sure the new procedure name begins with “sp_”.
- Review the *Transact-SQL User’s Guide* explanation of how to create and execute stored procedures.

Stored Procedure Code

```
CREATE PROC sp_vdevno AS
SELECT vdevno = low/16777216, name, status from
sysdevices
where cntrltype = 0
```

7 SQL Server Performance Issues

This chapter presents material about some issues relating to SQL Server performance.

Using the *showplan* Command

This section explains how to use and interpret the **showplan** command to better understand and utilize the SQL Server query optimizer.

When you send a SQL statement to the Sybase SQL Server, the request first goes to a cost-based query optimizer whose job it is to find the most efficient data access path to fulfill the request. To do this, the optimizer examines such information as:

- The structure of any indices defined on the table(s) involved
- The distribution of data within the indices
- The number of rows in the table(s) involved
- The number of data pages used by the table(s) involved
- The amount of data cache SQL Server is currently using
- The access path that would require the least amount of I/O and, therefore, would be the fastest

Once an optimal access path is calculated, it is stored in a query plan within the procedure cache. For more information about query plans, refer to Chapter 5, “Stored Procedure Processing and Management.”

The **showplan** allows you to view the plan the optimizer has chosen and to follow each step that the optimizer took when joining tables, reading from an index or using one of several other methods to determine cost efficiency. To invoke the showplan command, enter:

```
1> set showplan on
2> go
```

This command causes SQL Server to display query plan information for every SQL statement executed within the scope of the SQL Server session.

Since the determination of a query plan is performed independently from the actual data retrieval or modification, it is possible to examine and tune query plans without actually executing the SQL statement. This can be accomplished by instructing SQL Server not to execute any SQL statements via the following command:

```
1> set noexec on
2> go
```

➤ ***Note***

Issue **noexec** after **showplan** or the **set showplan** command will not execute.

For more information about executing the **showplan** command, refer to the *System Administration Guide*.

➤ ***Note***

The **showplan** command does not function within stored procedures or triggers. However, if you set it to on and then execute a stored procedure or a command that fires a trigger, you can see the procedure or trigger output.

Use the following examples to analyze a query plan. In all cases, examples use the *pubs* database provided with each SQL Server release.

Interpreting *showplan* Output

The output of the **showplan** command consists of many different types of statements, depending on the details of the access path that is being used. The following sections describe some of the more common statements.

STEP *n*

This statement is added to the **showplan** output for every query, where *n* is an integer, beginning with 1. For some queries, SQL Server cannot effectively retrieve the results in a single step, and must break the query plan into several steps. For example, if a query includes a **group by** clause, the query needs to be broken into at least two steps: one to select the qualifying rows from the table and another to group them.

The following query demonstrates a single-step query and its **showplan** output:

```
1> select au_lname, au_fname from authors
2> where city = "Oakland"
3> go
```

```
STEP 1
The type of query is SELECT
FROM TABLE
authors
Nested iteration
Table Scan
```

A multiple-step example is shown in the next section.

The Type of Query Is SELECT (into a Worktable)

This **showplan** statement indicates that SQL Server needs to insert some of the query results into an intermediate worktable and, later in the query processing, select the values from that table. This is most often seen with a query which involves a **group by** clause, as the results are first put into a worktable, and then the qualifying rows in the worktable are grouped based on the given column in the **group by** clause.

The following query returns a list of cities and indicates the number of authors who live in each city. The query plan is composed of two steps: the first step selects the rows into a worktable, and the second step retrieves the grouped rows from the worktable.

```
1> select city, total_authors = count (*)
2> from authors group by city
3> go
```

```
STEP 1
The type of query is SELECT (into a worktable)
GROUP BY
Vector Aggregate
FROM TABLE
authors
Nested iteration
Table Scan
TO TABLE
Worktable
```

```
STEP 2
The type of query is SELECT
FROM TABLE
Worktable
Nested iteration
Table Scan
```

The Type of Query Is *query_type*

This statement describes the type of query for each step. For most user queries, the value for *query_ type* is **select**, **insert**, **update**, or **delete**. If **showplan** is turned on while other commands are issued, the *query_ type* reflects the command that was issued. The following two examples show output for different queries or commands:

```
1> create table Mytab (col1 int)
2> go
```

```
STEP 1
The type of query is CREATE TABLE
```

```
1> insert publishers
2> values ("9904", "NewPubs", "Nome", "AL")
3> go
```

```
STEP 1
The type of query is INSERT
The update mode is direct
Table Scan
TO TABLE
publishers
```

The Update Mode Is Deferred

There are two methods or, modes, that SQL Server can use to perform update operations such as **insert**, **delete**, **update**, and **select into.** These methods are called **deferred update** and **direct update**. When the deferred method is used, the changes are applied to all rows of the table by making log records in the transaction log to reflect the old and new value of the column(s) being modified (in the case of **update** operations), or the values that will be inserted or deleted (in the case of **insert** and **delete**).

When all log records have been constructed, the changes are applied to the data pages. This method generates more log records than a direct update, but it has the advantage of allowing commands to execute which may cascade changes throughout a table. For example, consider a table that has a column *col1* with a unique index on it and data values numbered consecutively from 1 to 100 in that column. Execute an **update** statement to increase the value in each row by one:

```
1> update Mytable set col1 = col1 + 1
2> go
```

```
STEP 1
The type of query is UPDATE
The update mode is deferred
FROM TABLE
Mytable
Nested iteration
Table scan
TO TABLE
Mytable
```

Consider the consequences of starting at the first row in the table, and updating each row until the end of the table. This violates the unique index. First, updating the first row (which has an initial value of 1) to 2 would cause an error, since 2 already exists in the table. Second, by updating the second row or any row in the table except the last one does the same.

Deferred updates avoid unique index violations. The log records are created to show the new values for each row, the existing rows are deleted and new values are inserted. In the following example, the table *authors* has no clustered index or unique index:

```
1> insert authors select * from authors
2> go
```

```
STEP 1
The type of query is INSERT
The update mode is deferred
FROM TABLE
authors
Nested iteration
Table Scan
TO TABLE
authors
```

Because the table does not have a clustered index, new rows are added at the end of the table. The query processor distinguishes between existing rows now in the table (before the **insert** command) from the rows to be inserted, thus avoiding the continuous loop of selecting a row, inserting it at the end of the table, reselecting the row just inserted and reinserting it. The deferred insertion method first creates the log records to show all currently existing values in the table. Then SQL Server rereads those log records to insert the rows into the table.

The Update Mode Is Direct

Whenever possible, SQL Server tries to directly apply updates to tables, since this is faster and creates fewer log records than the deferred method. Depending on the type of command, one or more criteria must be met in order for SQL Server to perform the update using the direct method. The criteria are as follows:

- **insert** – Using the direct method, the table into which the rows are being inserted cannot be a table which is being read from in the same command. The second query example in the previous section demonstrates this, where the rows are being inserted into the same table in which they are being selected from. In addition, if rows are being inserted into the target table, and one or more of the target table's columns appear in the **where** clause of the query, then the deferred method, rather than the direct method, will be used.
- **select into** – When a table is being populated with data by means of a **select into** command, the direct method will always be used to insert the new rows.
- **delete** – For the direct update method to be used for **delete**, the query optimizer must be able to determine that either zero or one row qualifies for the **delete**. The only way to verify this is to check that one unique index exists on the table, which is qualified in the **where** clause of the **delete** command, and the target table is not joined with any other table(s).
- **update** – For the direct update method to be used for **update** commands, the same criteria apply as for **delete**: a unique index must exist so that the query optimizer can determine that no more than one row qualifies for the update, and the only table in the **update** command is the target table to update. Also, all updated columns must be fixed-length datatypes, not variable-length datatypes. Note that any column that allows null values is internally stored by SQL Server as a variable-length datatype column.

```
1> delete from authors
2> where au_id = "172-32-1176"
3> go
```

```
STEP 1
The type of query is DELETE
The update mode is direct
FROM TABLE
authors
Nested iteration
Using Clustered Index
TO TABLE
authors
```

```
1> update titles set type = 'popular_comp'
2> where title_id = "BU2075"
3> go
```

```
STEP 1
The type of query is UPDATE
The update mode is direct
FROM TABLE
titles
Nested iteration
Using Clustered Index
TO TABLE
titles
```

```
1> update titles set price = $5.99
2> where title_id = "BU2075"
3> go
```

```
STEP 1
The type of query is UPDATE
The update mode is deferred
FROM TABLE
titles
Nested iteration
Using Clustered Index
TO TABLE
titles
```

Note that the only difference between the second and third example queries is the column of the table which is updated. In the second query, the direct update method is used, whereas in the third query, the deferred method is used. This difference occurs because of the datatype of the column being updated: the *titles.type* column is defined as “char(12) NOT NULL” where the *titles.price* column is defined as “money NULL”. Since the *titles.price* column is not a fixed-length datatype, the direct method cannot be used.

GROUP BY

This statement appears in the **showplan** output for any query that contains a **group by** clause. Queries that contain a **group by** clause are always two-step queries: the first step selects the qualifying rows into a table and groups them; the second step returns the rows from the table as seen in the following example:

```
1> select type, avg (advance), sum(ytd_sales)
2> from titles group by type
3> go
```

```
STEP 1
The type of query is SELECT (into a worktable)
GROUP BY
Vector Aggregate
FROM TABLE
titles
Nested iteration
Table Scan
TO TABLE
Worktable

STEP 2
The type of query is SELECT
FROM TABLE
Worktable
Nested iteration
Table Scan
```

For more information on **group by**, refer to Chapter 8, “The tempdb Database.”

Scalar Aggregate

Transact-SQL includes the aggregate functions **avg**, **count**, **max**, **min**, and **sum**. Whenever you use an aggregate function in a **select** statement that does not include a **group by** clause, the result is a single value, regardless of whether it operates on all table rows or on a subset of the rows defined in the **where** clause. When an aggregate function produces a single value, the function is called a **scalar aggregate** and **showplan** lists it that way as seen in the following example:

```
1> select avg(advance), sum(ytd_sales) from titles
2> where type = "business"
3> go
```

```
STEP 1
The type of query is SELECT
Scalar aggregate
FROM TABLE
titles
Nested iteration
Table scan

STEP 2
The type of query is SELECT
Table Scan
```

showplan considers this a two-step query, which is similar to the **group by** output. Since the query contains a scalar aggregate which will return a single value, SQL Server keeps a “variable” internally to store the result of the aggregate function. It can be thought of as a temporary storage space to keep a running total of the aggregate function as the qualifying rows from the table are evaluated. After all rows are evaluated from the table in step 1, the final value of the variable is selected in step 2 to return the scalar aggregate result.

Vector Aggregates

When a **group by** clause is used in a query that also includes an aggregate function, the aggregate function produces a value for each group. These values are called **vector aggregates**. The **vector aggregate** statement from **showplan** indicates that the query includes a vector aggregate. The following example query includes a vector aggregate:

```
1> select title_id, avg (qty) from sales
2> group by title_id
3> go

STEP 1
The type of query is SELECT (into a worktable)
GROUP BY
Vector Aggregate
FROM TABLE
sales
Nested iteration
Table Scan
TO TABLE
Worktable
```

```
STEP 2
The type of query is SELECT
FROM TABLE
worktable
Nested iteration
Table Scan
```

from table Statement

This **showplan** output shows the table from which the query reads. In most queries, the **from table** is followed by the table's name. In other cases, it may show that it is selecting from a worktable. The significant fact is that the **from table** output show the query optimizer's order for joining tables. The order in which the tables are listed is the order in which the tables are joined. This order often differs from the order in which tables are listed in the query's **from** or **where** clauses. The reason for this is that the query optimizer checks many join orders for the tables and picks the order that uses the fewest I/Os.

```
1> select authors.au_id, au_fname, au_lname
2> from authors, titleauthor, titles
3> where authors.au_id = titlesauthor.au_id
4> and titleauthor.title_id = titles.title_id
5> and titles.type = "psychology"
6> go
```

```
STEP 1
The type of query is SELECT
FROM TABLE
TITLES
Nested iteration
Table Scan
FROM TABLE
TITLEAUTHOR
Nested iteration
Table Scan
FROM TABLE
authors
Nested iteration
Table Scan
```

This query illustrates the join order that the query optimizer chose for the tables, which is not the order listed in either the **from** or **where** clauses. By examining the order of the **from table** statements, it can be seen that the qualifying rows from the titles table are first located with the search clause **titles.type = "psychology"**. Those rows are then joined with the *titleauthor* table using the join clause **titleauthor.title_id =**

titles.title_id. Finally, the *titleauthor* table is joined with the *authors* table to retrieve the desired columns using the join clause **authors.au_id = titleauthor.au_id**.

to table Statement

When you issue a command that tries to modify one or more table rows, such as **insert**, **delete**, **update**, or **select into**, the **to table** statement shows the target table that is being modified. If the operation requires an intermediate step and inserts the rows into a worktable, the **to table** statement names the worktable instead of the user table.

```
1> insert sales
2> values ("8042", "QA973", "7/15/94", 7,
3> "Net 30", "PC1035")
4> go
```

```
STEP 1
The type of query is INSERT
The update mode is direct
TO TABLE
sales
```

```
1> update publishers
2> set city = "Los Angeles"
3> where pub_id = "1389"
4> go
```

```
STEP 1
The type of query is UPDATE
The update mode is deferred
FROM TABLE
publishers
Nested iteration
Using Clustered Index
TO TABLE
publishers
```

Note that the **showplan** for the second query indicates that the *publishers* table is used for both **from table** and **to table**. With **update** operations, the query optimizer must first read the table containing the row(s) to be updated, resulting in the **from table** statement, and then must modify the row(s), resulting in the **to table** statement.

Worktable

For some queries, such as those that require ordered or grouped output, the query optimizer creates its own temporary table called a worktable. The worktable holds all the intermediate results of the query where they are ordered and/or grouped, and then the final select is done. When all results are returned, the table is dropped automatically. The *tempdb* database holds all temporary tables so the System Administrator may need to increase the size of that database to accommodate very large worktables. For more information about worktables, refer to Chapter 8, “The tempdb Database.”

Since the query optimizer creates these worktables for its own internal use, the worktable names are not listed in the *tempdb..sysobjects* table.

Nested Iteration

The nested iteration is the default technique used to join table and return rows from a table. It indicates that the query optimizer uses one or more sets of loops to read a table and fetch a row, qualify the row based on the search criteria given in the **where** clause, return the row to the front end, and then loop again for the next row. The following example shows the query optimizer doing nested iterations through each of the tables in the join:

```
1> select title_id, tile
2> from titles, publishers
3> where titles.pub_id = publishers.pub_id
4> and publishers.pub_id = '1389'
5> go
```

```
STEP 1
The type of query is SELECT
FROM TABLE
publishers
Nested iteration
Using clustered index
FROM TABLE
titles
Nested iteration
Table Scan
```

Table Scan

This **showplan** statement identifies the method used to fetch the physical result rows from the given table. When the **table scan** method is used, execution begins with the first row on the table. Then each row is fetched and compared with the conditions set in the **where** clause, then returned as valid data if the conditions are met. No matter how many rows qualify, every row in the table must be checked, and this causes problems if the table is large (the scan has a high I/O overhead). If a table has one or more indexes on it, the query optimizer may still choose a table scan instead of reading the index. The following query shows a typical table scan:

```
1> select au_lname, au_fname
2> from authors
3> go
```

```
STEP 1
The type of query is SELECT
FROM TABLE
authors
Nested iteration
Table Scan
```

Cursors and Index Selection

This section describes the index selection for cursors.

If a cursor is marked for read only or is not updatable (**group by, aggregate**, or **order by** for language and SQL Server cursors), then any index can be used. For example:

```
1> declare c2 cursor for select au_lname,
2> au_fname from authors
3> group by au_lname, au_fname
4> go
```

```
STEP 1
The type of query is SELECT (into a worktable).
GROUP BY
Vector Aggregate
FROM TABLE
authors
Nested iteration
Index : aunmind
TO TABLE
Worktable
```

```
STEP 2
The type of query is SELECT.
FROM TABLE
Worktable
Nested iteration
Table Scan
```

If a cursor is updatable but not specified as **for update**, then any unique index will be chosen over a table scan or non-unique index, although either of those is possible if no unique indexes exist.

If a cursor is declared **for update** but without a **for update of** list, then a unique index must be chosen. Error 311 will be raised if there is no unique index.

If there is a **for update of** list, then only a unique index which does not include any column in the list as part of its key can be chosen. Error 311 will be raised if there is no unique index.

For the purposes of index selection for cursors, an index which contains an identity column as part of its key is considered unique even if it is not declared as unique.

When only a unique index can be chosen, SQL Server will choose the best unique index. For example, if there is no index which covers the query (all columns being returned are in the index) and no applicable search argument terms exist, then SQL Server will use the least expensive index, not the first one that it finds.

Force index adds additional complexity to this:

- If there is no **for update** clause, then the cursor becomes read only.
- If there is a **for update** clause and you specify a non-unique index or a unique index whose key includes 1 or more columns in the **for update of** list, then SQL Server will issue a warning (cursor may be closed on update). This warning will only be issued once.

An index is forced through the **from** clause: **from tablename (indexid)**. For example:

```
1> select au_id from authors(0)
2> go
```

This forces SQL Server to use index ID 0 as an access path to the table's data, that is, to table scan.

8 The *tempdb* Database

This chapter describes the SQL Server temporary database, *tempdb*, including managing the database, error recovery, and performance tuning information.

How *tempdb* Works

The *tempdb* database is a work space shared by all databases and users on a SQL Server. Space in *tempdb* is allocated either explicitly, as requested by the user, or implicitly during certain operations.

Implicit Usage

SQL Server uses *tempdb* as an area to store worktables that include the results of queries that involve sorting and **select into** and reformatting strategies for the query optimizer (as explained below). SQL Server does not use *tempdb* when you **create index** on other databases.

Worktables Created by Queries

There are five types of queries that may implicitly create worktables in *tempdb*:

- **select distinct**
- **select ...order by**
- **select ... group by**
- **select ... where not exists**
- **select ... where ... or ...**

These types are described in the following sections.

select distinct

A worktable is created to sort the results of a **select distinct** so that no duplicates are returned.

select ... order by

A worktable is created for queries using **order by** only when the table (or tables) involved in the query does not have a clustered index on the column(s) listed in the **order by** clause.

If a nonclustered index exists on all of the columns in an **order by** list, it will be considered by the query optimizer (for SQL Server releases 4.9.1 and later) as an alternative to creating a worktable and sorting it. For example:

```
1> select au_lname, au_fname from authors
2> order by au_lname, au_fname
3> go
```

In the above example, there is a nonclustered index on both of the columns in the **order by** clause. The query optimizer will weigh the cost of sorting by the nonclustered index on *au_lname*, *au_fname* against the cost of creating a worktable for the two columns and sorting the worktable.

If the estimated cost of using the nonclustered index is cheaper (in terms of the number of I/Os), a worktable will not be used.

select ... group by

Use of the **group by** function always forces creation of a worktable for intermediate query results.

select ... where not exists

Queries that use the **not exists** clause and a join need a worktable for intermediate query results. These results are processed through a call to the **group by** function. In the example below, the table *foo* was created by **select into ... from** *authors.* Note where the worktable is created and used:

```
1> select au_lname from authors a
2> where not exists
3> (select 1 from foo b
4> where b.au_lname = a.au_lname)
5> go
```

The join above forces use of the worktable.

The select below is into a worktable.

```
STEP 1

The type of query is SELECT (into a worktable).
GROUP BY
FROM TABLE
authors
Nested iteration
Index : aunmind
TO TABLE
Worktable

STEP 2
The type of query is SELECT (into a worktable).
GROUP BY
Vector Aggregate
FROM TABLE
authors
Nested iteration
Index : aunmind
FROM TABLE
foo
Nested iteration
Table Scan
TO TABLE
Worktable

STEP 3
The type of query is SELECT.
FROM TABLE
authors
Nested iteration
Index : aunmind
FROM TABLE
Worktable
Nested iteration
Using Clustered Index
 au_lname
 ----------------------------------------

(0 rows affected)
```

select... group by, select... not exists and **select where ... or** all cause a clustered index on the worktable to be created for internal table join processing. This index exists only until the query is completed.

If a query has been qualified to limit the number of rows returned, the worktable generated by that query can be significantly smaller than the original table. Additionally, the worktable may require fewer columns than are in the table selected from, since only those columns which are used in the query are required by the worktable. However, one additional column may be added: for example, the aggregate column for a **group by**, such as a **count**.

Creation of a clustered index on the worktable requires that the available space in *tempdb* be equal to 120 percent to 150 percent of the size of the worktable. This space will be deallocated when the sort completes. For more information, see the *System Administration Guide*.

You can check to see if your query creates a worktable by entering the following command:

```
1> set showplan on
2> go
```

and then running the query. If the output contains the word “Worktable,” then your query is creating a worktable in *tempdb*. For example:

```
1> select * from authors
2> order by au_lname
3> go
```

```
STEP 1
The type of query is INSERT
The update mode is direct.
Worktable created for ORDER BY
```

(Space in *tempdb* will be needed.)

The diagnostics available to monitor space used by worktables are limited, because worktables have no entries in the system catalogs.

SQL Server releases 10.0 and later include a threshold manager, which monitors the space used in a database. The SYBASE System Administrator can set this up either to dump a transaction log when the available space falls below a certain threshold or to print messages to the error log before space runs out. For more information, see “Managing Free Space with Thresholds” in the *System Administration Guide*.

Worktables Created by Reformatting Strategies

If the tables involved in a query have no useful indexes for the purpose of the query, the optimizer makes restricted copies in *tempdb* of some of the tables involved in the query and creates an index or indexes on them in order to speed data access and query completion. This is called **reformatting**.

Worktables Created by select into

Every time you use **select into**, a worktable is created to provide temporary storage for the selected data before it is inserted into the destination table.

Temporary Tables and Worktables Created by System Procedures

Some system stored procedures create temporary tables and worktables. As a result, the execution of most system stored procedures requires space in *tempdb*.

Explicit Usage

Users can create their own temporary tables in *tempdb*, either by preceding the table's name in a **create table** statement with a pound sign (#) or by specifying the path *tempdb..tablename*.

Temporary Tables Created with "#" Prefix

Temporary tables created with the pound sign (#) are accessible only by the SQL Server session that created them. Users running other sessions cannot read or write to them. These tables are dropped automatically when the session that created them terminates.

The first 13 bytes of the table name, including the pound sign, must be unique for the current session. Long temporary table names are truncated to 13 characters with the pound sign. Short table names are padded to 13 characters with underscores (_). An attempt to create a second temporary table with the same name in the same session will raise Error 2714:

```
There is already an object named '%.*s' in the
database.
```

SQL Server assigns a 17-byte number suffix to the names of temporary tables created with the pound sign prefix. You can see this suffix by selecting from *tempdb..sysobjects*:

```
1> select name from tempdb..sysobjects
2> where name like '%temp%'
3> go
```

```
name
------------------------------
#temp________00000010011744201
```

Tables with this suffix can only be accessed from the session in which they were created (for SQL Server releases 4.8 and later).

Be aware that even using the full table name will not enable a user to drop or access a temporary table created by another process using this method. The naming convention is provided here simply to explain what the user will see on a **select from sysobjects** in *tempdb*.

You cannot change the names of temporary tables with the # prefix by using **sp_rename**. Temporary tables created within a stored procedure are dropped when the stored procedure completes execution or drops the tables by giving the **drop table** command. Tables created outside of a stored procedure are dropped when the user logs out of SQL Server or explicitly drops them with the **drop table** command.

Temporary Tables Created with the “tempdb..” Prefix

More “permanent” temporary tables can be created, which are accessible to some other SQL Server processes by creating them explicitly in *tempdb* without the # prefix. SQL Server adds no suffixes to temporary tables created this way. The table exists either until the owner drops it using **drop table**, or until the SQL Server is rebooted. These tables obey the same rules as other user tables created in user databases except that they are dropped at SQL Server boot time.

Managing *tempdb*

There are a number of issues involved in managing *tempdb*, just as with any other database, though *tempdb* does have particular features that require planning for.

Sizing

There is no specific formula for the sizing of *tempdb*. You must determine its size based on a knowledge of your particular installation. Keep these issues in mind:

- How many users will be running concurrent queries or stored procedures that involve *tempdb* at any one time?

- Are there or will there be user-created temporary tables in *tempdb* and how large are they likely to be?
- How large are the worktables that might be involved in a join or **order by** clause?

A rule of thumb is this: the total size required in *tempdb* for worktables is the sum of the worktables required for every user on the SQL Server concurrently doing a query which requires a worktable. As mentioned above, space in *tempdb* can be monitored in 10.0 SQL Server with the use of thresholds.

In 4.9.x and later SQL Server, **dbcc checkalloc** is the best tool for checking consistency of database allocation. However, **dbcc checkalloc** does not lock the database from other processes, so diagnosis of potential problems with **dbcc checkalloc** may be misleading if it is run concurrently with other processes. For more information on how to more effectively run **dbcc checkalloc** in multi-user mode, see "How to Fix and Prevent Allocation Errors" on page 6-22.

For all worktables, the space required can be significantly less than the original table size, depending on qualifications in the query that limit the number of rows to be sorted. Only those columns used in the query are required by the worktable, though there may be an extra column (such as the result of an aggregate).

Error Recovery

Corruption in *tempdb* is rarely a problem, since the database is re-created each time SQL Server is booted. 605 and 821 errors in *tempdb* should clear on reboot of SQL Server. If you have repeated instances of corruption in *tempdb*, you may want to look for corruption in *model*, since *tempdb* is always built out of *model* on SQL Server start-up.

If you have increased the size of *tempdb* to the point where SQL Server cannot recover it, you can reduce the size of *tempdb* using the techniques shown in the following section.

Reducing the Size of *tempdb*

The *tempdb* (temporary) database provides storage for temporary tables and other temporary working storage needs. At some point, you may want to reduce the size of *tempdb*. The following procedure shows how to reduce *tempdb* to its default size (2MB of data and log on the master device).

◆ ***WARNING!***

Each time SQL Server is rebooted, the *model* database is copied to *tempdb*. Therefore, if the *model* database has been increased beyond its default size, do not reduce the size of *tempdb* so that it is smaller than *model*.

Reset *tempdb* to Default Size

Before proceeding, boot SQL Server in single-user mode to prevent another user from altering the database while you are manually updating *sysusages*. Refer to "How to Start SQL Server in Single-User Mode" on page 6-3 for instructions for doing this.

1. Log into SQL Server as the System Administrator:

   ```
   % isql -Usa -Sserver_name -Ppassword
   ```

2. Dump the *master* database in case something goes wrong and you need to restore from the backup:

   ```
   1> dump database master
   2> to "dump_device"
   3> go
   ```

   where *dump_device* is the name of the target dump device.

3. Save the output from the following **select** statements for information on key system tables, to aid in *master* database recovery if necessary:

   ```
   1> use master
   2> go

   1> select * from sysusages
   2> go

   1> select * from sysdevices
   2> go

   1> select * from sysdatabases
   2> go

   1> select * from syslogins
   2> go
   ```

◆ ***WARNING!***

This procedure should be used only on *tempdb*. It works because *tempdb* is rebuilt each time the system is rebooted. Use of this procedure on any other database will result in database corruption. Serious problems can result from an incorrectly modified system table.

4. Reconfigure SQL Server to allow changes to the system catalog:

   ```
   1> use master
   2> go

   1> sp_configure "allow updates", 1
   2> go

   1> reconfigure with override
   2> go
   ```

5. Display the current rows belonging to *tempdb* from *sysusages*, and note the number of rows affected:

   ```
   1> begin transaction
   2> go

   1> select * from sysusages
   2> where dbid = db_id('tempdb')
   3> go
   ```

   The **db_id** function returns the database ID number. In this case, the database ID for *tempdb* is returned.

6. Set the first 2MB of *tempdb* back to data and log in case they were separated:

   ```
   1> update sysusages
   2> set segmap = 7 where dbid = db_id('tempdb')
   3> and lstart = 0
   4> go
   ```

7. Delete all other rows belonging to *tempdb* from *sysusages*.The number of rows affected should be one less than the number of rows affected by the previous select.

   ```
   1> delete sysusages where dbid = db_id('tempdb')
   2> and lstart != 0
   3> go
   ```

8. Verify that *tempdb* has one entry that looks like this:

   ```
   1> select * from sysusages
   2> where dbid = db_id('tempdb')
   ```

```
dbid  segmap  lstart   size   vstart
   2       7       0   1024     2564
```

9. If the information is correct, go to step 10 to commit the transaction.

   If you see a problem, back out your changes by entering the following commands:

   ```
   1> rollback transaction
   2> go
   ```

   Do not continue with the procedure. Review the steps you performed to determine the cause of the problem.

10. Complete the transaction:

    ```
    1> commit transaction
    2> go
    ```

11. Reconfigure SQL Server to disallow changes to the system catalog (the normal state for SQL Server):

    ```
    1> sp_configure "allow updates", 0
    2> go
    ```

    ```
    1> reconfigure
    2> go
    ```

12. Immediately issue a **checkpoint** and shut down SQL Server (if you continue to run without rebooting, you will receive serious errors on *tempdb*):

    ```
    1> checkpoint
    2> go
    ```

    ```
    1> shutdown
    2> go
    ```

13. Restart SQL Server.

Verify and Alter *tempdb* on Desired Devices

Verify that *tempdb* has been correctly reset, and alter the database as required to include any additional devices:

1. Log into SQL Server as the System Administrator:

   ```
   % isql -Usa -Sserver_name -Ppassword
   ```

2. Verify that *tempdb* has one 2MB fragment for data and log on the master device:

   ```
   1> sp_helpdb tempdb
   2> go
   ```

3. Alter *tempdb* as required to extend the database on to the desired devices. For example:

   ```
   1> alter database tempdb
   2> on device_name = device_size
   3> go
   ```

   Note that *device_size* is specified in megabytes.

4. Back up the *master* database again, in case you need to restore from this point:

   ```
   1> dump database master to "dump_device"
   2> go
   ```

   where *dump_device* is the name of the target dump device.

You can use **sp_logdevice** to place the transaction log on another device. The first 2MB of *tempdb* data and log must remain on the *master* device, but future log space allocations will be made on the device specified by **sp_logdevice**.

Log Management

Like every other database, *tempdb* has its own transaction log. The same information is logged in *tempdb* as is logged for user tables in any other user database. That is, any user-created temporary table will log all data and overhead information in *tempdb*.

The only exception to this rule are the worktables. Worktables are system-generated, temporary tables, which are generated for intermediate results for certain types of queries or certain query processing strategies. Because normal transaction and recovery issues do not apply, only the allocation records for these tables, and not the data itself, need to be logged in *tempdb*.

Since the transaction log in *tempdb* is never needed for up-to-the-minute recovery, SQL Server always treats it as though the **truncate log on chkpt** option were enabled, even if you turn off that option with **sp_dboption** (*tempdb* is unique in this regard). Therefore, the transaction log for *tempdb* will be truncated roughly every 60 seconds.

It is still possible, however, to run out of log space in *tempdb*. **truncate log on chkpt** will only truncate back to the last open transaction. If there is, for example, a transaction that has not been committed or aborted by the user who opened it, or one that has failed to close through backout errors (failure to clean up after a killed process or other error), the log may not clear enough to prevent 1105 errors.

Additionally, the checkpoint process itself takes time. If the log is near full, while the checkpoint process is writing out pages, the log may fill enough that the final checkpoint record, as it is written to the log, will raise the 1105 error. To avoid this, you must do just as you would for judging log size on any other database, but you must additionally take into account the number of users likely to be using queries that indirectly involve *tempdb* at any given time, and the number of users who use *tempdb* directly.

9 SQL Server Auditing

SQL Server releases 10.0 and later support auditing at the C2 level of security. This chapter describes some issues that may be of interest when using SQL Server Auditing.

Purpose of SQL Server Auditing

When auditing object accesses, the data that is read, modified, or deleted is not part of the audit record. The purpose of object access auditing at the C2 level is to detect **attempts** to access objects. The event that gets audited is the permission check. Therefore, the actual data is not relevant and is not part of the audit record.

Rolled-Back Updates

When an update is performed and then rolled back, the permissions check on the update is audited. Thus, a single audit record is generated to indicate that the attempt to update was either successful (permission to update granted) or unsuccessful (permission to update denied).

If permission was granted, whatever happens after that is not reflected in the audit trail. This includes the possibility that the update failed for some other reason (log full, for example), or that the update was eventually rolled back. Similarly, if the command failed prior to reaching the permission check (syntax error, for example), no audit record will be generated.

Auditing on Views and Tables

If auditing is enabled on a view and a table, and the view is based on that same table, an access through the view will generate one or two audit records, depending on whether or not the view and table owner are the same. If they are, then only the view access will generate an audit record, because only the view access will cause a permission check to be performed by SQL Server. If the view and table owners are different, then both the table and view accesses will generate audit records, because each access will cause a permission check to be performed. The same is true for stored procedures and the objects within them—that is, if auditing is enabled on a stored procedure and some objects (for example, views or tables) are referenced in the stored procedure, the “common ownership” rules apply here as well.

If auditing table accesses by a certain user (login) is enabled as well as auditing on a particular table accessed by that user, two audit records will be generated. One of the audit records will have the generic “table access” event code, while the other will have the “login table access” event code. Note that it is not possible to restrict the “login table access” auditing to a particular table.

The *sysaudits* Table

Writes to the *sysaudits* table are not logged. However, in order to maintain recoverability, the page allocations are logged.

If the *sysaudits* table fills up, when the audit process tries to insert the records that encounter the “table full” condition, the audit process will terminate. After this, any user process that encounters an auditable event will also terminate. The only exception is that a System Security Officer (SSO) can perform auditable tasks (some of the audit data associated with these tasks will be sent to the SQL Server error log). The SSO should free up some space in *sysaudits*, and then shut down and restart SQL Server. When the audit process has terminated abnormally, the SSO does have permission to shut down SQL Server.

When SQL Server terminates abnormally, the audit records in the audit queue are lost. Also, an additional 20 audit records that have been written to a page buffer but not yet flushed to disk may be lost.

◆ ***WARNING!***

Do not enable auditing unless you need it, as auditing can fill up *syslogs* in your *sybsecurity* database.

Effect of Roles on Auditing

Some commands require that a user be either a System Administrator (SA) **or** possess some other characteristic, such as Database Owner (DBO). Since an SA is a DBO wherever he or she goes, it is not known whether the user is the true DBO or an SA who becomes the DBO by default. Thus, if the user has the SA role enabled, auditing will occur even if that user is also truly the DBO. The audit record may be extraneous.

10 Disk Mirroring

This chapter discusses disk mirroring issues related to recovery.

Disk Mirroring Overview

This chapter discusses disk mirroring issues involving recovery. Use it with the *System Administration Guide*, which explains the uses of mirroring as well as the commands involved. The last section of this chapter contains instructions for using disk mirroring for other system administration tasks even when disk mirroring is not normally used.

Disk mirroring allows continued processing in the event of a single device failure. Recovery involving mirrored devices is usually handled automatically by SQL Server and no action is required by the user. When SQL Server encounters an I/O error, it disables mirroring and continues processing without interruption. Following are some scenarios that do not result in I/O errors and therefore do not disable mirroring:

- A particular block on a device is bad. SQL Server does not detect an I/O error and disable mirroring until the block is accessed.
- Data on a device is overwritten. Database corruption results, but mirroring is not disabled. For example, if a mirrored device is accidentally mounted as a UNIX file system, UNIX could potentially overwrite SYBASE data but SQL Server would not encounter an I/O error.
- Incorrect data is written to both the primary and secondary device.

Disk mirroring is not designed to detect or prevent database corruption. Because scenarios such as those mentioned above can cause corruption, you should regularly run consistency checks such as **dbcc checkalloc** and **dbcc checkdb** on all databases.

Disk Mirroring Commands and the *sysdevices* Table

Mirror recovery uses information stored in the system catalog table *sysdevices*. The *sysdevices* table is defined as follows:

Table 10-1: The sysdevices table

Column Name	Datatype	Length	Nulls
low	*int*	4	0
high	*int*	4	0
status	*smallint*	2	0
cntrltype	*smallint*	2	0
name	*sysname*	30	0
phyname	*varchar*	127	0
mirrorname	*varchar*	127	1

The *mirrorname* and *status* columns in this table give the most information for recovering mirrored devices. *mirrorname* is the full physical name of the secondary, or mirror, device. Note that the mirror device does not have its own row in *sysdevices*. The mirror device is listed in the primary device's row in *sysdevices* in the *mirrorname* column.

The **disk mirror** command initializes the mirror device. Do not execute a **disk init** on the secondary device before you execute a **disk mirror.**

If you use operating system files as database devices, the **disk mirror** command creates the mirror device file. Always mirror raw devices to raw devices, and operating system files to files.

status Column Bit Masks

The *status* column is a bit mask indicating the type of device and the mirror status. The bit representations for mirroring are discussed below.

32 - Serial Writes

Writes to the disk devices are serial. The write operation must be completed on the primary device before it is begun on the secondary device. Refer to the *System Administration Guide* for more details.

64 - Device Mirrored

This bit indicates that the device is set up as a mirrored device. Note that this bit remains on even when mirroring is temporarily disabled.

128 - Reads Mirrored

This bit indicates that SQL Server attempts to determine which side of the mirrored device (primary or secondary) will yield better performance on a read operation. This bit is always on when the device is mirrored.

256 - Half Mirror Only

This bit is only used if the Mirror Enable bit (512) is off, indicating that mirroring has been temporarily disabled via the **disk unmirror** command. If this bit is on, only the secondary device will be used. If this bit is off, only the primary device will be used.

512 - Mirror Enabled

If this bit is on, then mirroring is active on this device and the Half Mirror bit (256) is not examined. If this bit is off, mirroring is still considered **active** even though only one of the primary or secondary devices is currently being used.

2048 - Mirror Disabled

If this bit is on, then mirroring is not active on this device.

Following are some examples of disk mirroring commands and their effects on the appropriate columns of *sysdevices*:

Example 1

```
1> disk init name = "test",
2> physname = "/usr/sybase/test.dat",
3> size=5120, vdevno=3
4> go
```

```
name  phyname               mirrorname  status
test  /usr/sybase/test.dat  NULL        2
```

Status 2 indicates that the device is a physical disk. Since the device mirrored bit (64) is off and the *mirrorname* column is null, this device is not mirrored.

Example 2

```
1> disk mirror name = "test",
2> mirror = "/usr/sybase/test.mir"
3> go
```

```
name  phyname               mirrorname            status
test  /usr/sybase/test.dat  /usr/sybase/test.mir  738
```

Status 738 indicates that mirroring is currently active (512) on this device. Reads are mirrored (128), and writes are mirrored (64) and serial (32). The device is a physical disk (2).

Example 3

```
1> disk unmirror name = "test",
2> side = secondary, mode = retain
3> go
```

```
name  phyname               mirrorname            status
test  /usr/sybase/test.dat  /usr/sybase/test.mir  226
```

Status 226 indicates that mirroring has been disabled (512 bit off) and that only the primary device is used (256 bit off). Reads are mirrored (128), and writes are mirrored (64) and serial (32). The device is a physical disk (2).

Example 4

```
1> disk remirror name = "test"
2> go
```

```
name  phyname               mirrorname            status
test  /usr/sybase/test.dat  /usr/sybase/test.mir  738
```

Status 738 indicates that mirroring is currently active (512) on this device. Reads are mirrored (128), and writes are mirrored (64) and serial (32). The device is a physical disk (2).

Example 5

```
1> disk unmirror name = "test",
2> side = primary, mode = retain
3> go
```

```
name  phyname               mirrorname           status
test  /usr/sybase/test.dat  /usr/sybase/test.mir 482
```

Status 482 indicates that mirroring has been disabled (512 bit off) and that only the secondary device is used (256). Reads are mirrored (128), and writes are mirrored (64) and serial (32). The device is a physical disk (2).

Example 6

```
1> disk remirror name = "test"
2> go
```

```
name  phyname               mirrorname            status
test  /usr/sybase/test.dat  /usr/sybase/test.mir  738
```

Status 738 indicates that mirroring is currently active (512) on this device. Reads are mirrored (128), and writes are mirrored (64) and serial (32). The device is a physical disk (2).

Example 7

```
1> disk unmirror name = "test", side = primary,
2> mode = remove
3> go
```

```
name  phyname               mirrorname            status
test  /usr/sybase/test.mir  NULL                  2
```

Status 2 indicates that the device is a physical device. Since the *mirrorname* column is null, mirroring is not enabled on this device.

◆ ***WARNING!***

If you encounter disk mirroring behavior that does not match the behavior described in this chapter, the *sysdevices* table might contain an incorrect status. If the status is incorrect, contact Sybase Technical Support. The status may have to be changed before you can continue.

Failure Scenarios Involving Disk Mirroring

Following are some failure scenarios involving disk mirroring that illustrate the expected behavior of SQL Server and the effects on *sysdevices*.

I/O Error Detected on Primary Device

If an I/O error is detected on the primary device, SQL Server disables mirroring. *sysdevices* reflects information for the device as if a **disk unmirror** with **side = primary** and **mode = retain** was issued (the 512 bit is turned off and the 256 bit is turned on). SQL Server does this automatically, and the error log displays kernel messages such as:

```
kernel: udunmirror:i/o error on primary device'primary_device'
kernel: DataServer i/o to the device will be disabled
kernel: udunmirror: failing over to 'secondary_device'
```

Once the device is replaced or becomes accessible to SQL Server, you can make the device available again. See “Finishing Up After the Device Is Repaired” on page 10-7.

I/O Error Detected on Secondary Device

If an I/O error is detected on the secondary device, SQL Server disables mirroring and *sysdevices* reflects information for the device as if a **disk unmirror** with **side = secondary** and **mode = retain** was issued (the 512 bit is turned off and the 256 bit is turned off). SQL Server does this automatically. The error log displays kernel messages such as:

```
kernel: udunmirror:i/o error on secondary device
'/usr/u/sybase/test.mir'
kernel: DataServer i/o to the device will be disabled
```

Once the device is replaced or becomes accessible to SQL Server, you can make the device available again. See “Finishing Up After the Device Is Repaired” on page 10-7.

Changing Permissions on Device Fails to Trigger I/O Failure

Do not change permissions to test disk mirroring.

Some Sybase customers try to test disk mirroring by changing permissions on one device, hoping to trigger I/O failure and therefore unmirror the other device. But the UNIX operating system does not check permissions on a device after it is opened, so the I/O failure will not occur until the next time the device is started.

Device Is Going Bad but No I/O Error Is Detected by SQL Server

The operating system administrator may notice that a device must be replaced or repaired before SQL Server detects it. If so, you might need to disable mirroring to allow repair work to be done on the disk. To do this, use the **disk unmirror** command. Refer to the *SQL Server Reference Manual* for a description of this command. Use **side** to indicate the device being removed from the SYBASE system for repair or replacement, and use **mode** to indicate whether this is permanent or temporary.

Once the device is replaced or becomes accessible to SQL Server, you can make the device available again. See “Finishing Up After the Device Is Repaired”.

Finishing Up After the Device Is Repaired

Remirror the device after it is repaired. Use the sections below, depending on whether you replaced or repaired a device, and whether you use operating system files as devices.

After Repairing or Replacing a Device

If the name of the device brought back to the system changes, use the next section for instructions. For example, a device with physical name */dev/rid001h* could have been replaced by */dev/rid002h.* In this case, see “Using Operating System Files for SYBASE Devices” on page 10-8.

➤ ***Note***

You may be able to replace the device with a device that has a different name by allowing updates to the system catalog, changing the *mirrorname* or *phyname* in *sysdevices*, and issuing a **disk remirror** command. Because the physical device names are also stored in internal SYBASE data structures, you must restart SQL Server before this technique works. Contact Sybase Technical Support if you need assistance with this method.

Once a device has been replaced or repaired, use the **disk remirror** command to make the device accessible to SQL Server. This command can be issued even if the device has been replaced by a new device. Refer to the *SQL Server Reference Manual* for a description of the **disk remirror** command.

You can use the **disk remirror** command if SQL Server automatically disabled mirroring or if you issued a **disk unmirror with mode = retain** command. If you previously issued a **disk unmirror with mode = remove** command, you can only reinstate disk mirroring by issuing the **disk mirror** command.

Using Operating System Files for SYBASE Devices

When using operating system files as SYBASE devices, the **disk remirror** command does not create the file for a device. Therefore, execute commands similar to the following if the file does not exist:

```
1> disk unmirror name=device_name,
2> side={primary|secondary}, mode=remove
3> go
```

```
1> disk mirror name=device_name,
2> mirror=full_mirror_name
3> go
```

Refer to the *SQL Server Reference Manual* for a description of the **disk unmirror** and **disk mirror** commands. Use **side** to indicate the device that was previously temporarily disabled.

➤ ***Note***

On a UNIX system, you can also use the operating system command **touch** to create the file and then issue a **disk remirror** command as shown in the previous section.

Special Considerations for Master Device

When a master device is mirrored, the *phyname* in *sysdevices* changes from “the_master_device” to the actual physical name. In addition, the mirror name is included in *sysdevices* and is hard coded on the master device in the configuration block. For this reason, if either the primary or secondary device name changes, you must unmirror the device with **mode = remove** and then use **disk mirror** to specify the new device name.

Because the configuration block stores the physical names, you must rebuild it with the **buildmaster -r** command before this technique will work. Contact Sybase Technical Support if you need help with this method.

◆ ***WARNING!***

If you have followed all of the procedures in this document and SQL Server does not appear to access the appropriate master device when starting, call Sybase Technical Support for assistance. It might be necessary to use buildmaster -r to clear all information regarding mirroring in the configuration block.

Starting SQL Server with a Mirrored Device

If you mirror the master device after you have installed SQL Server rather than during installation, edit the runserver file and add the **-r** flag (on UNIX) to indicate the physical name of the mirror device. For example:

On UNIX:

```
#!/bin/csh -f
# Server name: TEST
# dslisten port: 4011
# master name: /dev/rid001d
# master size: 5120
setenv DSLISTEN TEST
exec /sybase/bin/dataserver -dprimary_device -rsecondary_device
     -e/sybase/install/errorlog_TEST
```

➤ ***Note***

The **exec** command must be on one line.

On Digital OpenVMS:

```
! servername: TEST
! dslisten port: 4011
define/nolog/process/dslisten TEST
server:== $sybase_system:[sybase.bin]dataserver.exe
server /device=primary_device,-
    disk$mirror:secondary_device-
     /error=sybase_system:[sybase.install]error_log
```

On Novell, no runserver file is used. Use the **-r** flag on the file server console command line:

```
LOAD SQLSRVR -dSYS:\SYBASE\MASTR.DAT -rSYS:\SYBASE\MASTER.MIR
```

Simply including the **-r** flag in the runserver file (or at the command line for Novell) does not mirror the device, that must be done according to normal system administration procedures. Including the **-r** flag just tells SQL Server to start up with mirroring.

Once you have edited it, the runserver file does not need to be edited again if one of the devices fails.

You must remove the **-r** option if the master device mirror is permanently disabled with **disk unmirror** and **mode = remove**. If the primary side is permanently disabled, edit the runserver file to include the new physical name (the old mirror name) for the **-d** option and to remove the **-r** option. In addition, if either the primary or secondary device names change, edit the runserver file to reflect those changes.

11 Novell NetWare Issues

This chapter discusses issues that may arise when using SQL Server in a Novell NetWare environment.

➤ ***Note***

This chapter applies to Novell NetWare versions 3.11, 3.12, 4.01, 4.02, and 4.10.

Novell Memory Issues

This section explains how to assess memory use on your NetWare file server and describes a memory limitation on pre-10.x SQL Servers.

Assessing Memory Use

Use the Novell NetWare MONITOR utility to check how much memory is available on the NetWare file server:

1. At the file server console, type:

   ```
   :load monitor
   ```

2. Select Resource Utilization.
3. Look at the percentage of cache buffers displayed. This number indicates how much of the total NetWare file server memory is available. This number should never fall below 30 percent of total memory.

   To achieve usage above 30 percent, you can do one of three things:

   - Add more memory
   - Use **sp_configure** to reduce the amount of memory allocated to the SQL Server
   - Unload other NLMs

4. Check the parameters for Minimum File Cache Buffers. To do this, type the **set** command and then select File Caching from the menu. The default value is 20. This parameter sets the limit past

which the file server stops giving file cache buffers to other processes.

16MB Memory Limit on Some Disk Adaptor and Network Boards

If you are running a SQL Server release that precedes 10.x and you have a network board or disk adaptor board that uses inline DMA or AT bus-mastering, you are limited to 16MB of NetWare file server memory. Using more memory may cause corruption. See your Novell *System Administration Manual* for more information.

Troubleshooting Tape Dump Problems

This section gives some preliminary troubleshooting suggestions for systems whose tape dumps appear not to be working.

Determine Whether Your Problem Is SYBASE-Specific

To determine whether your problem is SYBASE-specific, run the Novell NetWare SBACKUP utility. Because SBACKUP and SYBASE software use the same components needed to communicate to the tape device, if SBACKUP cannot dump to the tape drive, then SYBASE cannot dump to tape.

If SBACKUP fails, your problem is Novell-specific. See your Novell NetWare documentation for help.

If SBACKUP is successful but you still cannot make SYBASE tape dumps, your problem is SYBASE-specific. Read the section below entitled “Preliminary Steps” and then find the configuration that matches your site and follow the troubleshooting instructions.

Preliminary Steps

1. Check the *SYBASE SQL Server Release Bulletin* and make sure all instructions have been followed.
2. Verify that you have the current versions of the following Sybase-supplied NLM files:
   - *sys:/system/dibi/dibidai.nlm* (loaded automatically at time of dump)
   - *sys:/system/tapedai.dsk* (loaded by *dibidai.nlm*)

- *sys:/physaddr.dsk* (loaded by *dibidai.nlm*)

  You may be missing these files if there was a problem with installing, upgrading, or starting SQL Server.

  If *tapedai.dsk* and *physaddr.dsk* are present, check that they are not old versions of the files. To do this, unload *physaddr.dsk*—it should not unload without unloading *tapedai.dsk* first. Then reload *tapedai.dsk*—it should not load without loading *physaddr.dsk* first.

3. Type the following at the file server command line to check your search path:

   ```
   :search
   ```

   You should see the directory *sys:/system/dibi* in the search output. If it is not there, add the directory to your search path using your regular Novell operating system procedures.

4. Verify that the file *sys:/system/dibi/dibi2$dv.dat* has an entry similar to the following:

   ```
   "Tape_Drive_Name" dibidai
   ```

➤ ***Note***

Place a tab, not single spaces, between the tape drive name and “dibidai”.

5. Some programs, conflict with SYBASE because they take full control of the hardware and do not release it to other applications. Verify that no other third party backup software NLMs are loaded, by issuing the NetWare **modules** command at the file server console. If they are, unload them from memory.

6. Verify that a SYBASE dump device has been created, and has the proper controller number, by following these steps:

   1. Use **sp_helpdevice** to check that a dump device exists.
   2. Verify that the value for *controller* in the **sp_helpdevice** output is a number from 4 to 8 (inclusive). 3 is reserved for floppy drives.
   3. Check the size of the device to make sure it is large enough.
   4. If no dump device exists, see the *SQL Server Reference Manual* entry for **sp_addumpdevice** for information about creating a dump device.

For *NOVADIBI.NLM* Dumps

➤ ***Note***

If the PC has a network board or disk adaptor board that uses inline DMA or AT bus-mastering (for example, an ISA bus type), then *novadibi.nlm* does not work properly and SQL Server will not be able to dump to tape. Use the *dibidai.nlm* if possible.

1. If you are running a 3.x version of NetWare, verify that the following entry exists in the *startup.ncf* file:

   ```
   set reserved buffers below 16 meg = 150
   ```

2. Make sure that all NLMs provided by the tape drive manufacturer have been installed in the proper location.

➤ ***Note***

Sybase does not recommend one tape drive over another. The following Adaptec information is supplied for your information only and does not imply a preference for that brand of SCSI host adapter.

For Adaptec brand host adapters, there should be files in the DOS partition called ASPITRAN.DSK and AHA*model_number*.DSK (where *model_numbe*r is the model number of the controller board). These files are provided by the SCSI manufacturer. AHA1540 is for ISA, AHA1640 is for MCA (microchannel) and AHA1740 is for EISA.

You can automatically load the Adaptec brand SCSI driver's NLM by placing the following statement in the *startup.ncf* file:

```
:load AHAmodel_number.DSK port=330 DMA=5 IRQ=11
```

Make sure the port, DMA, and IRQ values (which may be different for your site) are also in the *sys:/system/dibi/dibi2$dv.da*t file in the *novadibi* line. Here is the proper syntax for the entry:

```
:load novadibi port=### dma=# irq=#
```

➤ ***Note***

For parameter information about other controllers, contact the manufacturer.

For *DIBIDAI.NLM* Dumps

Novell uses the DAI format to talk to tape drives, but SQL Server uses DIBI. The *DIBIDAI.NLM* module translates from DIBI to DAI.

The order of files to load is:

1. Load tape driver (*aha1540.dsk*).
2. Load *tapedai.dsk*.

dibidai.nlm will load automatically upon execution of the **dump database** command.

When You Perform a Tape Dump

After verifying all the information above, including the fact that current versions of *tapedai.dsk* and *physaddr.dsk* are already loaded, use a procedure similar to the following to perform a tape dump:

1. Type the following command at the file server console:

   ```
   :load console
   ```

2. Press Ctrl-Esc and choose System Console from the menu. Then type the following command:

   ```
   :LOAD ISQL -Usa -P
   ```

3. Once logged into the **isql** session, issue the **dump database** command to device *tape_drive_name*.

   You can toggle between four screens:

   - SYBASE console
   - SYBASE ISQL UTILITY (this is running the dump command.)
   - File Server System Console
   - SYBASE NETWARE SQL Server (displays error log)

4. In SQL Server releases 4.2.2 and later, the SYBASE **console** utility asks for block size. Enter the block size at the prompt.

➤ ***Note***

Once you have issued the **isql** dump command, watch the screen for messages. The SQL server error log gets informational messages as well as error messages. The file server console may or may not generate informational and error messages, so watch for these, too.

5. Test the backup tape by restoring it. One technique is to delete some rows from a test table, then load the dump back in and see if the rows have been restored.

Recommended List of NLMs

Before starting SQL Server, load the NetWare Loadable Modules (NLMs) that Sybase recommends for your version of NetWare:

Table 11-1: Recommended NLMs, by NetWare version

Netware Version	Recommended NLMs	
3.11	• clib	• spxddfix
	• mathlib	• spxfix2
	• mathlibc	• spxfsfix
	• a3112	• spxlisfx
	• after311	• xmdfix
	• streams	• patchman
	• spxs	• directfs
	• tli	
	• ipxs	
3.12	• clib	• tli
	• mathlib	• ipxs
	• mathlibc	• spxddfix
	• a3112	• pm312
	• after311	• directfs
	• streams	
	• spxs	

Sybase does not provide these modules. Download the latest version of NLMs from Novell's Compuserve forums.

Load Order for NLMs

Novell recommends loading NLMs in the following order:

1. LOAD STREAMS
2. LOAD CLIB

3. LOAD MATHLIBC
4. LOAD TLI
5. LOAD IPXS
6. LOAD SPXS
7. LOAD DIRECTFS
8. LOAD PATCHMAN
9. LOAD SPXFSFIX
10. LOAD SPXLISFX
11. LOAD SPXFIX2

The *.txt* files included in the files you download from Compuserve contain detailed information about each NLM.

Performance and Tuning Parameters

This section describes NetWare parameters that you can use to help tune the performance of your SQL Server.

Volume Block Size

Use the "Volume Block Size" parameter to free up memory for SQL Server's use. Increasing the volume block size reduces the size of the File Allocation Table (FAT), which leaves more memory available to your SQL Server.

➤ ***Note***

You can set this parameter only when creating a new volume.

Since SQL Server uses DFS (DirectFS.NLM), I/O is done in the "requested" number of sectors size (2K for SQL Server) rather than the volume block size. Therefore, this parameter does not directly affect I/O performance.

In non-dedicated environments, consider dedicating separate volumes to SQL Server device files with a larger block size (8K or 16K) while storing other files on other volumes with the appropriate block sizes for their characteristics.

Cache Buffer Size

The Cache Buffer Size parameter controls the block size of the cache buffer. While this parameter will not have significant performance impact, NetWare will not mount volumes whose volume block size is smaller than their cache buffer size.

Language Installation Problems

Missing French and German Localization Files

SQL Server 4.2 is missing French and German localization files. EBF 5264 fixes this problem.

Japanese Language Installation on NetWare 3.12

sybinit fails to install the Japanese language and character set modules in SQL Server 10.0.2 running on NetWare 3.12. EBF 5264 fixes this problem.

Questions and Answers

This section answers questions that may arise when you run SQL Server on the Novell platform.

Question

Why can’t I connect to my SQL Server?

Answer

There are a variety of reasons for connection failure. Here are some things to check:

- Make sure that the *DSLISTEN* parameter in *sybenv.dat* is set to the correct SQL Server name.
- Check that SQL Server is up by entering the following command from the console:

  ```
  display servers
  ```

- Check the error log to make sure that SQL Server is advertising on the correct port. Remember to use the **cperrlog** utility to make a copy of the error log for viewing.
- Check to make sure that the server name entries in *sybenv.dat* and the interfaces file match. Server names are case-sensitive.

Question

Why can't I load **isql**?

Answer

Your interfaces file may contain inaccurate information or be improperly formatted.

Check your interfaces file for the following:

- Blank lines.
- Network numbers that do not match the one NetWare is using. Your interfaces file must contain the address that the NetWare file server advertises, which is in the *autoexec.ncf* file.

  For SQL Server running on an SPX network, the IPX internal number is most critical. For SQL Servers running on a TCP/IP network, the IP number is critical.
- Always use **sybinst** in SQL Server 4.2.2 and **sybinit** in SQL Server 10.x to change the interfaces file.

Question

What do I do when my **disk init** command fails with an error message about lack of space?

Answer

Configure your NetWare environment to ensure that deleted files are actually purged from the system, allowing the space used by those files to be returned to the system. You can do this immediately by typing the **purge/all** command from the root directory of a client machine.

To set up your environment to perform automatic, immediate purges, put the following lines in your *sys:/system/autoexec.ncf* file:

```
set immediate purge of deleted files =on
```

Question

When exiting from the **isql** NLM, other utilities, or SQL Server using a **quit** command, this message appears:

```
<Press any key to close screen>
```

I want to unload utilities or the SQL Server automatically, but an operator must be present to close the screen by typing a key. How can I get around this?

Answer

As of release 4.2.2, SQL Server and all SYBASE utilities have a **silent exit** option that obviates the need for an operator to strike a key.

Invoke SQL Server or the utility using normal procedures, but include the **-k** flag in the command; for example:

```
:LOAD ISQL -k
```

◆ ***WARNING!***

The -k flag has the opposite effect in System 10. That is, if you start isql with the -k flag, the user will be prompted to press a key to close the screen.

Note that you must include the **-k** flag with each NLM you load, to get the silent exit functionality. For example, if you need silent exits for both SQL Server and **isql** NLMs, be sure to invoke each NLM with the **-k** flag.

Question

What can I do to prevent NetWare SQL Server from monopolizing my machine's CPU?

Answer

Invoke the **sqlsrvr** NLM with the **-P** flag.

Consider using the **-P** option **only** when you are running complex queries that involve large amounts of data and you experience the following problems:

- SQL Server is dropping existing connections
- The NetWare Console is hanging
- The NetWare system clock is running slowly

Here is the syntax:

```
sqlsrvr -Pnumber
```

The *number* parameter controls the frequency with which the SQL Server relinquishes the CPU to other NetWare processes. The lower the value of *number*, the more often SQL Server will relinquish the CPU. The default value of *number* is 3000.

◆ ***WARNING!***

Do not use the -P option without a number following it. This can cause your entire file server to freeze when processing very large joins.

Question

Suppose my *autoexec.ncf* contains incorrect syntax or invokes a malfunctioning executable. How do I suppress execution of the *autoexec.ncf* configuration file when starting up my NetWare server?

Answer

Start your NetWare server with the **-na** (no autoexec) option on the command line:

```
server -na
```

Question

When I attempt to dump a database, NetWare returns the following error:

```
DFSExpandFile Failed ...
```

What does this error message mean?

Answer

There is insufficient space on the volume to which you are dumping. Free additional space on the volume by decreasing the time the NetWare file server waits before it actually deletes a file. Enter the following commands from the console:

```
set file delete wait time=60 sec
set minimum file delete wait time=30 sec
```

Question

How can I control whether or not SQL Server creates a dynamic socket in situations in which the interfaces file is corrupted or missing?

Answer

The *USE_DEFAULT_SPX* parameter in *sybenv.dat* controls whether or not SQL Server will create a dynamic socket.

If *USE_DEFAULT_SPX* is set to TRUE, SQL Server will dynamically generate and allocate a network address using SAP (NetWare Service Advertising Protocol) if no server information is found in the interfaces file. This means that applications that expect SQL Server to be listening on a particular socket may not be able to connect to the server. Applications must use the NetWare bindery in order to connect to a dynamic socket.

Question

Why do I see two network handler entries in my **sp_who** display?

Answer

Your SQL Server supports both the SPX and TCP protocols, and there are two entries in your interfaces file.

Question

Why is my database dump to tape failing with a segmentation fault?

Answer

This problem occurs only in 10.x SQL Servers. Your dump tape may write protected. Before performing a database dump to tape, make sure that the tape is not write protected.

Question

Can my Backup Server and SQL Server share a port number?

Answer

No. **sybinit** shows a default port number for a backup server that is the same as that for SQL Server.

Do not accept the default; enter the appropriate port number (in hexadecimal):

Protocol	SQL Server Port	Backup Server Port
SPX	0x83bd	0x83be
TCP	0x1000	0x1001

Question

Why is there a slight delay when my SQL Server boots on a large system?

Answer

SQL Server must allocate and then deallocate memory. For example, depending on the platform, a SQL Server with 16MB of memory may take up to one minute to initialize.

Question

Does SQL Server directly support NetWare Version 4.01 Directory Services?

Answer

No. If you are using NetWare Version 4.01, install NetWare Version 4.01 Directory Services with the Bindery Emulation mode.

Question

Can I reload SQL Server 10.x into the existing installation directory?

Answer

No. You must first remove the existing installation directory or load in another location. Otherwise, the load will fail because there are read-only files in the directory that cannot be updated.

Index