System Administration Guide
Volume 2

Adaptive Server Enterprise

12.5

DOCUMENT ID: 31644-01-1250-01

LAST REVISED: May 2001

Contents

CHAPTER 15 Overview of Disk Resource Issues

This chapter discusses some basic issues that determine how you allocate and use disk resources with Adaptive Server.

Topics covered in this chapter include:

Many Adaptive Server defaults are set to reasonable values for aspects of storage management, such as where databases, tables, and indexes are placed and how much space is allocated for each one. Responsibility for storage allocation and management is often centralized, and usually, the System Administrator has ultimate control over the allocation of disk resources to Adaptive Server and the physical placement of databases, tables, and indexes on those resources.

Device allocation and object placement

When configuring a new system, the System Administrator must consider several issues that have a direct impact on the number and size of disk resources required. These device allocation issues refer to commands and procedures that add disk resources to Adaptive Server. Device allocation topics are described in the chapters shown in Table 15-1.

Table 15-1: Device allocation topics

Task	Chapter
Initialize and allocate a default pool of database devices.	Chapter 16, “Initializing Database Devices”
Mirror database devices for recovery.	Chapter 17, “Mirroring Database Devices”

After the initial disk resources have been allocated to Adaptive Server, the System Administrator, Database Owner, and object owners should consider how to place databases and database objects on specific database devices. These object placement issues determine where database objects reside on your system and whether or not the objects share devices. Object placement tasks are discussed throughout this manual, including the chapters shown in Table 15-2.

Table 15-2: Object placement topics

Task	Chapter
Place databases on specific database devices.	Chapter 21, “Creating and Managing User Databases”
Place tables and indexes on specific database devices.	Chapter 23, “Creating and Using Segments”

Do not consider allocating devices separately from object placement. For example, if you decide that a particular table must reside on a dedicated pair of devices, you must first allocate those devices to Adaptive Server. The remaining sections in this chapter provide an overview that spans both device allocation and object placement issues, providing pointers to chapters where appropriate.

Commands for managing disk resources

Table 15-3 lists the major commands a System Administrator uses to allocate disk resources to Adaptive Server and provides references to the chapters that discuss those commands.

Table 15-3: Commands for allocating disk resources

Command	Task	Chapter

`disk init`
`name = "`*`dev_name`*`"`
`physname = "`*`phys_name`*`"...`
	Makes a physical device available to a particular Adaptive Server. Assigns a database device name (*dev_name*) that is used to identify the device in other Adaptive Server commands.	Chapter 16, "Initializing Database Devices"

`sp_deviceattr` *`logicalname, optname, optvalue`*	Changes the *dsync* setting of an existing database device file	Chapter 16, "Initializing Database Devices"
`sp_diskdefault "`*`dev_name`*`"...`	Adds *dev_name* to the general pool of default database space.	Chapter 16, "Initializing Database Devices"

`disk mirror`
`name = "`*`dev_name`*`"`
`mirror = "`*`phys_name`*`"...`
	Mirrors a database device on a specific physical device.	Chapter 17, "Mirroring Database Devices"

Table 15-4 lists the commands used in object placement. For information about how object placement affects performance, see Chapter 5, "Controlling Physical Data Placement," in the *Performance and Tuning Guide*.

Table 15-4: Commands for placing objects on disk resources

Command	Task	Chapter

`create database...on` *`dev_name`*
or
`alter database...on` *`dev_name`*
	Makes database devices available to a particular Adaptive Server database. The log on clause to create database places the database's logs on a particular database device.	Chapter 21, "Creating and Managing User Databases"

`create database...`
or
`alter database...`
	When used without the on *dev_name* clause, these commands allocate space on the default database devices.	Chapter 21, "Creating and Managing User Databases"

`sp_addsegment` *`seg_name,dbname, devname`*
and
`sp_extendsegment` *`seg_name, dbname, devname`*
	Creates a segment, a named collection of space, from the devices available to a particular database.	Chapter 23, "Creating and Using Segments"

`create table...on` *`seg_name`*
or
`create index...on` *`seg_name`*
	Creates database objects, placing them on a specific segment of the database's assigned disk space.	Chapter 23, "Creating and Using Segments"

Command	Task	Chapter
`create table...` or `create index...`	When used without on *seg_name*, tables and indexes occupy the general pool of space allocated to the database (the default devices).	Chapter 23, "Creating and Using Segments"

Considerations in storage management decisions

The System Administrator must make many decisions regarding the physical allocation of space to Adaptive Server databases. The major considerations in these choices are:

- **Recovery** – disk mirroring and maintaining logs on a separate physical device provide two mechanisms for full recovery in the event of physical disk crashes.
- **Performance** – for tables or databases where speed of disk reads and writes is crucial, properly placing database objects on physical devices yields performance improvements. Disk mirroring slows the speed of disk writes.

Recovery

Recovery is the key motivation for using several disk devices. Nonstop recovery can be accomplished by mirroring database devices. Full recovery can also be ensured by storing a database's log on a separate physical device.

Keeping logs on a separate device

Unless a database device is mirrored, full recovery requires that a database's transaction log be stored on a different device from the actual data (including indexes) of a database. In the event of a hard disk crash, you can create an up-to-date database by loading a dump of the database and then applying the log records that were safely stored on another device. See Chapter 21, "Creating and Managing User Databases," for information about the log on clause of create database.

Mirroring

Nonstop recovery in the event of a hard disk crash is guaranteed by of mirroring all Adaptive Server devices to a separate physical disk. Chapter 17, "Mirroring Database Devices," describes the process of mirroring devices.

Performance

You can improve system performance by placing logs and database objects on separate devices:

- Placing a table on one hard disk and nonclustered indexes on another ensures that physical reads and writes are faster, since the work is split between two disk drives.
- Splitting large tables across two disks can improve performance, particularly for multi-user applications.
- When log and data share devices, user log cache buffering of transaction log records is disabled.
- Partitioning provides multiple insertion points for a heap table, adds a degree of parallelism to systems configured to perform parallel query processing, and makes it possible to distribute a table's I/O across multiple database devices.

See Chapter 5, "Controlling Physical Data Placement," in the *Performance and Tuning Guide* for a detailed discussion of how object placement affects performance.

Status and defaults at installation time

You can find instructions for installing Adaptive Server in the installation documentation for your platform. The installation program and scripts initialize the master device and set up the master, model, sybsystemprocs, sybsecurity, and temporary databases for you.

When you install Adaptive Server, the system databases, system defined segments, and database devices are organized as follows:

- The master, model, and tempdb databases are installed on the master device.
- The sybsystemprocs database is installed on a device that you specified.
- Three segments are created in each database: system, default, and logsegment.
- The master device is the default storage device for all user-created databases.

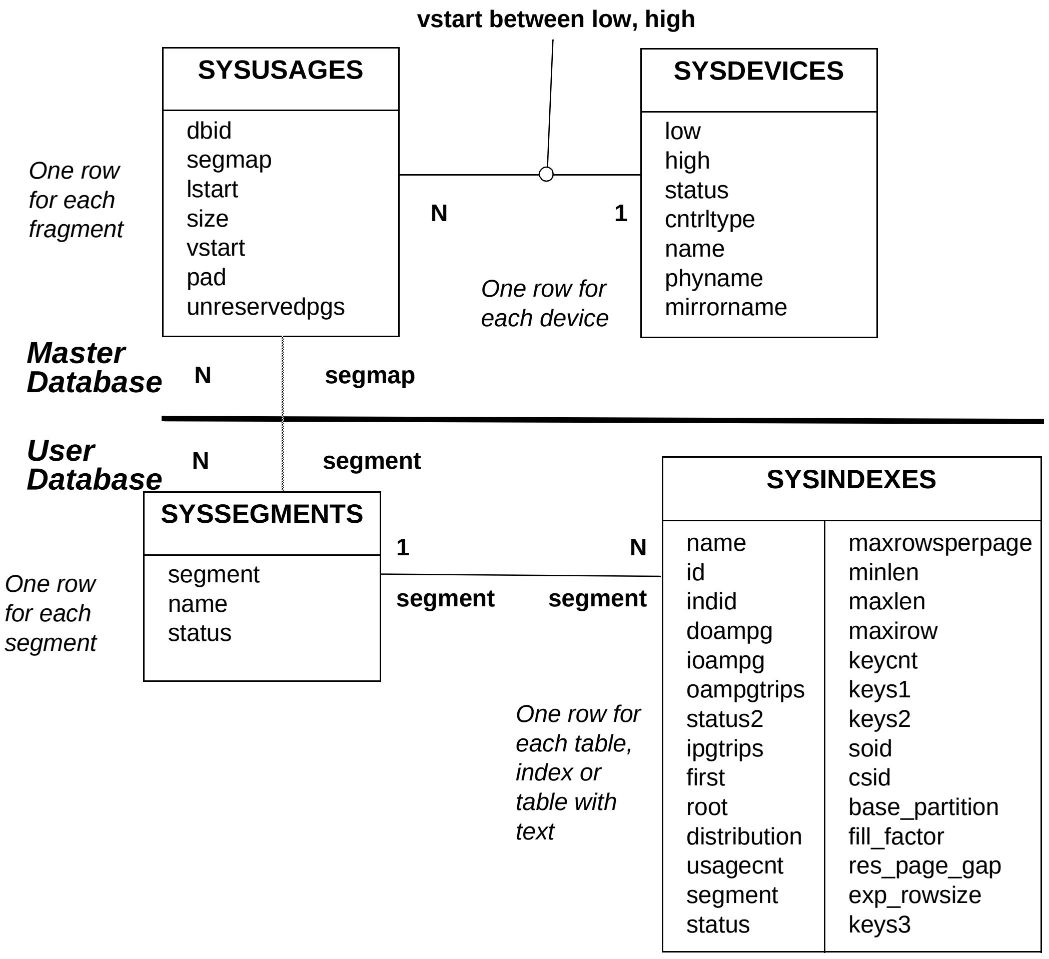
Note After initializing new devices for default storage, remove the master device from the default storage area with sp_diskdefault. Do not store user databases and objects on the master device. See "Designating default devices" on page 549 for more information.

- If you install the audit database, sybsecurity, it is located on its own device.

System tables that manage storage

Two system tables in the master database and two more in each user database track the placement of databases, tables (including the transaction log table, syslogs), and indexes. The relationship between the tables is illustrated in Figure 15-1.

Figure 15-1: System tables that manage storage



The *sysdevices* table

The sysdevices table in the master database contains one row for each **database device** and may contain a row for each dump device (tape, disk, or operating system file) available to Adaptive Server.

The disk init command adds entries for database devices to master..sysdevices. Dump devices, added with the system procedure sp_addumpdevice, are discussed in Chapter 26, “Developing a Backup and Recovery Plan.”

sysdevices stores two names for each device:

- A *logical name* or *device name*, used in all subsequent storage-management commands, is stored in the name column of sysdevices. This is usually a user-friendly name, perhaps indicating the planned use for the device, for example “logdev” or “userdbdev.”
- The *physical name* is the actual operating system name of the device. You use this name only in the disk init command; after that, all Adaptive Server data storage commands use the logical name.

You place a database or transaction log on one or more devices by specifying the logical name of the device in the create database or alter database statement. The log on clause to create database places a database’s transaction log on a separate device to ensure full recoverability. The log device must also have an entry in sysdevices before you can use log on.

A database can reside on one or more devices, and a device can store one or more databases. See Chapter 21, “Creating and Managing User Databases,” for information about creating databases on specific database devices.

The *sysusages* table

The sysusages table in the master database keeps track of all of the space that you assign to all Adaptive Server databases.

create database and alter database allocate new space to the database by adding a row to sysusages for each database device or device fragment. When you allocate only a portion of the space on a device with create or alter database, that portion is called a *fragment*.

The system procedures sp_addsegment, sp_dropsegment, and sp_extendsegment change the segmap column in sysusages for the device that is mapped or unmapped to a segment. Chapter 23, “Creating and Using Segments,” discusses these procedures in detail.

The *syssegments* table

The syssegments table, one in each database, lists the segments in a database. A **segment** is a collection of the database devices and/or fragments available to a particular database. Tables and indexes can be assigned to a particular segment, and therefore to a particular physical device, or can span a set of physical devices.

create database makes default entries in syssegments. The system procedures sp_addsegment and sp_dropsegment add and remove entries from syssegments.

The *sysindexes* table

The sysindexes table lists each table and index and the segment where each table, clustered index, nonclustered index, and chain of text pages is stored. It also lists other information such as the max_rows_per_page setting for the table or index.

The create table, create index, and alter table commands create new rows in sysindexes. Partitioning a table changes the function of sysindexes entries for the table, as described in Chapter 5, “Controlling Physical Data Placement,” in the *Performance and Tuning Guide*.

CHAPTER 16 Initializing Database Devices

This chapter explains how to initialize database devices and how to assign devices to the default pool of devices.

Topics covered in this chapter include:

What are database devices?

A database device stores the objects that make up databases. The term *device* does not necessarily refer to a distinct physical device: it can refer to any piece of a disk (such as a disk partition) or a file in the file system that is used to store databases and their objects.

Each database device or file must be prepared and made known to Adaptive Server before it can be used for database storage. This process is called **initialization**.

After a database device has been initialized, it can be:

- Allocated to the default pool of devices for the create and alter database commands
- Assigned to the pool of space available to a user database
- Assigned to a user database and used to store one or more database objects
- Assigned to store a database’s transaction logs

Using the *disk init* command

A System Administrator initializes new database devices with the disk init command, which:

- Maps the specified physical disk device or operating system file to a *database device* name
- Lists the new device in master..sysdevices
- Prepares the device for database storage

Note Before you run disk init, see the installation documentation for your platform for information about choosing a database device and preparing it for use with Adaptive Server. You may want to repartition the disks on your computer to provide maximum performance for your Sybase databases.

---

disk init divides the database devices into **allocation unit**s. The size of the allocation unit depends on which logical page size your server is configured for (2, 4, 8, or 16K). In each allocation unit, the disk init command initializes the first page as the allocation page, which will contain information about the database (if any) that resides on the allocation unit.

Warning! After you run the disk init command, dump the master database. This makes recovery easier and safer in case master is damaged. See Chapter 28, "Restoring the System Databases."

---

disk init syntax

The syntax of disk init is:

```
disk init
    name = "device_name" ,
    physname = "physicalname" ,
    [vdevno = virtual_device_number , ]
    size = size_of_device
    [, vstart = virtual_address ,
        cntrltype = controller_number]
    [, dsync = {true | false}]
```

disk init examples

On UNIX:

```
disk init
  name = "user_disk",
  physname = "/dev/rxy1a",
  size = “10M”
```

On Windows NT:

```
disk init
  name = "user_disk",
  physname = "d:\devices\userdisk.dat",
  size = “10M”
```

Specifying a logical device name with *disk init*

The *device_name* must be a valid identifier. This name is used in the create database and alter database commands, and in the system procedures that manage segments. The logical device name is known only to Adaptive Server, not to the operating system on which the server runs.

Specifying a physical device name with *disk init*

The *physicalname* of the database device gives the name of a raw disk partition (UNIX), foreign device, or the name of an operating system file. On PC platforms, you typically use operating system file names for *physicalname*.

Choosing a device number for *disk init*

Adaptive Server automatically specifies the next available identifying number for the database device. This is the virtual device number (vdevno). You do not have to specify this number when you issue the disk init command.

If you choose to manually select the vdevno, it must be unique among the devices used by Adaptive Server. Device number 0 represents the master device. The highest number must be one less than the number of database devices for which your system is configured. For example, for a system with a default configuration of 10 devices, the legal device numbers are 1–9. To see the configuration value for your system, execute sp_configure "number of devices" and check the run value:

```
                          sp_configure "number of devices"
Parameter name  Default  Memory Used  Config Value  Run Value
-------------- -------  -----------  ------------  ----------
number of devices    10           0           10          10
```

To see the numbers already in use for vdevno, look in the device_number column of the report from sp_helpdevice, or use the following query to list all the device numbers currently in use:

```
select distinct low/16777216
    from sysdevices
    order by low
```

Adaptive Server is originally configured for 10 devices. You may be limited to a smaller number of devices by operating system constraints. See the discussion of sp_configure, which is used to change configuration parameters, in Chapter 5, “Setting Configuration Parameters.”

Specifying the device size with *disk init*

You can use the following unit specifiers to indicate the size of the device: ‘k’ or ‘K’ indicate kilobytes, ‘m’ or ‘M’ indicate megabytes and ‘g’ or ‘G’ indicate gigabytes. Although it is optional, Sybase recommends that you always include the unit specifier in both the disk init and create database commands to avoid confusion in the actual number of pages allocated. You must enclose the unit specifier in single or double quotes.

The following apply to the syntax for disk init:

- You can specify the size as a float, but it is rounded down to the nearest whole value. For example, if you specify a size of 3.75G, it is rounded down to 3G.
- If you do not specify a size:
  - disk init and disk reinit use the basic disk page size of 2K.

- The size argument for create database and alter database is in terms of megabytes of disk piece. This value is converted to the number of logical page size that with which the master device was built.
- Minimum size of a database. You cannot alter the size of a database device after running disk init.
- If you are planning to use the new device for the creation of a new database, the minimum size depends on the logical page size used by the server, described in Table 16-1:

Table 16-1: Minimum database sizes

Logical page size	Minimum database size
2K	2 Megabytes
4K	4 Megabytes
8K	8 Megabytes
16K	16 Megabytes

If you are initializing a database device for a transaction log or for storing small tables or indexes on a segment, the size can be as small as 512 blocks (1MB).

If you are initializing a raw device, determine the size of the device from your operating system, as described in the the installation documentation for your platform. Use the total size available, up to the maximum for your platform. After you have initialized the disk for use by Adaptive Server, you cannot use any space on the disk for any other purpose.

disk init uses size to compute the value for the high virtual page number in sysdevices.high.

Warning! If the physical device does not contain the number of blocks specified by the size parameter, the disk init command fails. If you use the optional vstart parameter, the physical device must contain the sum of the blocks specified by both the vstart and size parameters, or the command fails.

Specifying the *dsync* setting with *disk init* (optional)

For devices initialized on UNIX operating system files, the dsync setting controls whether or not writes to those files are buffered. When the dsync setting is on, Adaptive Server opens a database device file using the UNIX dsync flag. The dsync flag ensures that writes to the device file occur directly on the physical storage media, and Adaptive Server can recover data on the device in the event of a system failure.

When dsync is off, writes to the device file may be buffered by the UNIX file system, and the recovery of data on the device cannot be ensured. The dsync setting should be turned off only when data integrity is not required, or when the System Administrator requires performance and behavior similar to earlier Adaptive Server versions.

Note The dsync setting is ignored for devices initialized on raw partitions, and for devices initialized on Windows NT files. In both cases, writes to the database device take place directly to the physical media.

Performance implications of *dsync*

The use of the dsync setting with database device files incurs the following performance trade-offs:

- HP-UX and Digital UNIX do not support asynchronous I/O on operating system files. If database device files on these platforms use the dsync option, then the Adaptive Server engine writing to the device file will block until the write operation completes. This can cause poor performance during update operations.
- When dsync is on, write operations to database device files may be slower compared to previous versions of Adaptive Server (where dsync is not supported). This is because Adaptive Server must write data to disk instead of simply copying cached data to the UNIX file system buffer.

  In cases where highest write performance is required (but data integrity after a system failure is not required) turning dsync off yields device file performance similar to earlier Adaptive Server versions. For example, you may consider storing tempdb on a dedicated device file with dsync disabled, if performance is not acceptable while using dsync.

- Response time for read operations is generally better for devices stored on UNIX operating system files as compared to devices stored on raw partitions. Data from device files can benefit from the UNIX file system cache as well as the Adaptive Server cache, and more reads may take place without requiring physical disk access.
- The disk init command takes longer to complete with previous Adaptive Server versions, because the required disk space is allocated during device initialization.

Limitations and restrictions of *dsync*

The following limitations and restrictions apply to using the dsync setting:

- dsync is always set to “true” for the master device file. You cannot change the dsync setting for the master device. If you attempt to turn dsync off for the master device, Adaptive Server displays a warning message.
- If you change a device file’s dsync setting using the sp_deviceattr procedure, you must reboot Adaptive Server before the change takes affect.
- When you upgrade from an Adaptive Server prior to version 12.x, dsync is set to “true” for the master device file only. You must use the sp_deviceattr procedure to change the dsync setting for any other device files.
- Adaptive Server ignores the dsync setting for database devices stored on raw partitions. Writes to devices stored on raw partitions are always done directly to the physical media.
- Adaptive Server also ignores the dsync setting for database devices stored on Windows NT operating system files. Adaptive Server on Windows NT automatically uses a capability similar to dsync for all database device files.

Other optional parameters for *disk init*

vstart is the starting virtual address, or the offset, for Adaptive Server to begin using the database device. vstart accepts the following optional unit specifiers: k or K (kilobytes), m or M (megabytes), and g or G (gigabytes). The size of the offset depends on how you enter the value for vstart.

- If you do not specify a unit size, vstart uses 2K pages for its starting address. For example, if you specify `vstart = 13`, Adaptive Server uses 13 * 2K pages as the offset for the starting address.
- If you specify a unit value, vstart uses this as the starting address. For example, if you specify `vstart = "13M"`, Adaptive Server sets the starting address offset at 13 megabytes.

The default value (and usually the preferred value) of vstart is 0. If the specified device does not have the sum of vstart + size blocks available, the disk init command fails.

The optional cntrltype keyword specifies the disk controller. Its default value is 0. Reset it only if instructed to do so.

Note To perform disk initialization, the user who started Adaptive Server must have the appropriate operating system permissions on the device that is being initialized.

Getting information about devices

The system procedure sp_helpdevice provides information about the devices in the sysdevices table.

When used without a device name, sp_helpdevice lists all the devices available on Adaptive Server. When used with a device name, it lists information about that device. Here, sp_helpdevice is used to report information about the master device:

```
                          sp_helpdevice master
device_name  physical_name  description
-----------  -------------- ----------------------------------------
master       d_master       special, default disk, physical disk, 20 MB


status      cntrltype    device_number     low      high
------      ----------   -------------     ------   -------
3           0            0                 0        9999
```

Each row in master..sysdevices describes:

- A dump device (tape, disk, or file) to be used for backing up databases, or

- A database device to be used for database storage.

The initial contents of sysdevices are operating-system-dependent. Entries in sysdevices usually include:

- One for the master device
- One for the sybsystemprocs database, which you can use to store additional databases such as pubs2 and sybsyntax, or for user databases and logs
- Two for tape dump devices

If you installed auditing, there will also be a separate device for sybsecurity.

The low and high fields represent the page numbers that have been assigned to the device. For dump devices, they represent the media capacity of the device.

The status field in sysdevices is a bitmap that indicates the type of device, whether a disk device will be used as a default storage device when users issue a create or alter database command without specifying a database device, disk mirroring information, and dsync settings. The status bits and their meanings are listed in Table 16-2:

Table 16-2: Status bits in sysdevices

Bit	Meaning
1	Default disk (may be used by any create or alter database command that does not specify a location)
2	Physical disk
4	Logical disk (not used)
8	Skip header (used with tape dump devices)
16	Dump device
32	Serial writes
64	Device mirrored
128	Reads mirrored
256	Secondary mirror side only
512	Mirror enabled
2048	Used internally; set after disk unmirror, side = retain
4096	Primary device needs to be unmirrored (used internally)
8192	Secondary device needs to be unmirrored (used internally)
16384	UNIX file device uses dsync setting (writes occur directly to physical media)

For more information about dump devices and sp_addumpdevice, see Chapter 26, “Developing a Backup and Recovery Plan.”

Dropping devices

To drop database and dump devices, use sp_dropdevice. The syntax is:

```
sp_dropdevice logicalname
```

You cannot drop a device that is in use by a database. You must drop the database first.

sp_dropdevice removes the device name from sysdevices. sp_dropdevice does not remove an operating system file: it only makes the file inaccessible to Adaptive Server. You must use operating system commands to delete a file after using sp_dropdevice.

Designating default devices

To create a pool of default database devices to be used by all Adaptive Server users for creating databases, use sp_diskdefault after the devices are initialized. sp_diskdefault marks these devices in sysdevices as default devices. Whenever users create (or alter) databases without specifying a database device, new disk space is allocated from the pool of default disk space.

The syntax for sp_diskdefault is:

sp_diskdefault *logicalname*, {defaulton | defaultoff}

You are most likely to use the defaultoff option to remove the master device from the pool of default space:

```
sp_diskdefault master, defaultoff
```

The following command makes sprocdev, the device that holds the sybsystemprocs database, a default device:

```
sp_diskdefault sprocdev, defaulton
```

Adaptive Server can have multiple default devices. They are used in the order in which they appear in the sysdevices table (that is, alphabetical order). When the first default device is filled, the second default device is used, and so on.

Note After initializing a set of database devices, you may want to assign them to specific databases or database objects rather than adding them to the default pool of devices. For example, you may want to make sure a table never grows beyond the size of a particular device.

Choosing default and nondefault devices

sp_diskdefault lets you plan space usage carefully for performance and recovery, while allowing users to create or alter databases.

Make sure these devices are *not* default devices:

- The master device (use sp_diskdefault to set defaultoff after adding user devices)
- The device for sybsecurity
- Any device intended solely for logs

- Devices where high-performance databases reside, perhaps using segments

You can use the device that holds sybsystemprocs for other user databases.

Note If you are using disk mirroring or segments, you should exercise caution in deciding which devices you add to the default list with sp_diskdefault. In most cases, devices that are to be mirrored or databases that will contain objects placed on segments should allocate devices specifically, rather than being made part of default storage.

CHAPTER 17 Mirroring Database Devices

This chapter describes creating and administering disk mirrors.

Topics convered in this chapter include:

What’s disk mirroring?

Disk mirroring can provide nonstop recovery in the event of media failure. The disk mirror command causes an Adaptive Server database device to be duplicated, that is, all writes to the device are copied to a separate physical device. If one device fails, the other contains an up-to-date copy of all transactions.

When a read or write to a mirrored device fails, Adaptive Server “unmirrors” the bad device and displays error messages. Adaptive Server continues to run unmirrored.

Deciding what to mirror

When deciding to mirror a device, you must weigh such factors as the costs of system downtime, possible reduction in performance, and the cost of storage media. Reviewing these issues will help you decide what to mirror—just the transaction logs, all devices on a server, or selected devices.

Note You cannot mirror a dump device.

You should mirror all default database devices so that you are protected if a create or alter database command affects a database device in the default list.

In addition to mirroring user database devices, you should always put their transaction logs on a separate database device. You can also mirror the database device used for transaction logs for even greater protection.

To put a database's transaction log (that is, the system table syslogs) on a different device than the one on which the rest of the database is stored, name the database device and the log device when you create the database. You can also use alter database to add a second device and then run the system procedure sp_logdevice.

Here are three examples that involve different cost and performance trade-offs:

- *Speed of recovery* – you can achieve nonstop recovery when the master and user databases (including logs) are mirrored and can recover without the need to reload transaction logs.
- *Storage space* – immediate recovery requires full redundancy (all databases and logs mirrored), which consumes disk space.
- *Impact on performance* – Mirroring the user databases (as shown in Figure 17-2 and Figure 17-3) increases the time needed to write transactions to both disks.

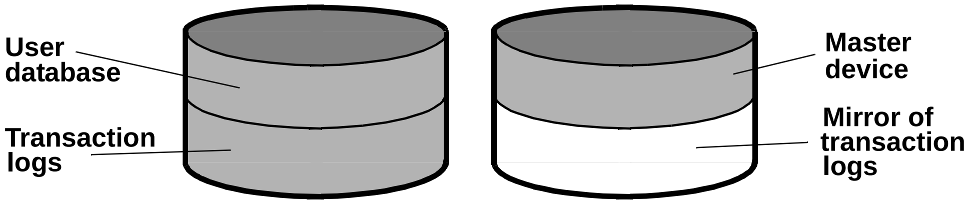
Mirroring using minimal physical disk space

Figure 17-1 illustrates the "minimum guaranteed configuration" for database recovery in case of hardware failure. The master device and a mirror of the user database transaction log are stored in separate partitions on one physical disk. The other disk stores the user database and its transaction log in two separate disk partitions.

If the disk with the user database fails, you can restore the user database on a new disk from your backups and the mirrored transaction log.

If the disk with the master device fails, you can restore the master device from a database dump of the master database and remirror the user database's transaction log.

Figure 17-1: Disk mirroring using minimal physical disk space



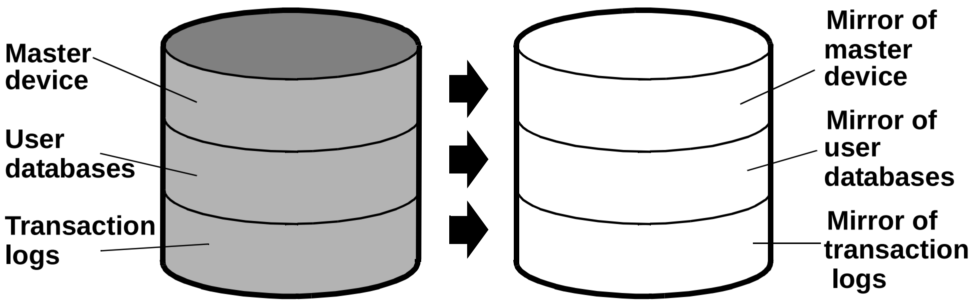
This configuration minimizes the amount of disk storage required. It provides for full recovery, even if the disk storing the user database and transaction log is damaged, because the mirror of the transaction log ensures full recovery. However, this configuration does not provide nonstop recovery because the master and user databases are not being mirrored and must be recovered from backups.

Mirroring for nonstop recovery

Figure 17-2 represents another mirror configuration. In this case, the master device, user databases, and transaction log are all stored on different partitions of the same physical device and are all mirrored to a second physical device.

The configuration in Figure 17-2 provides nonstop recovery from hardware failure. Working copies of the master and user databases and log on the primary disk are all being mirrored, and failure of either disk will not interrupt Adaptive Server users.

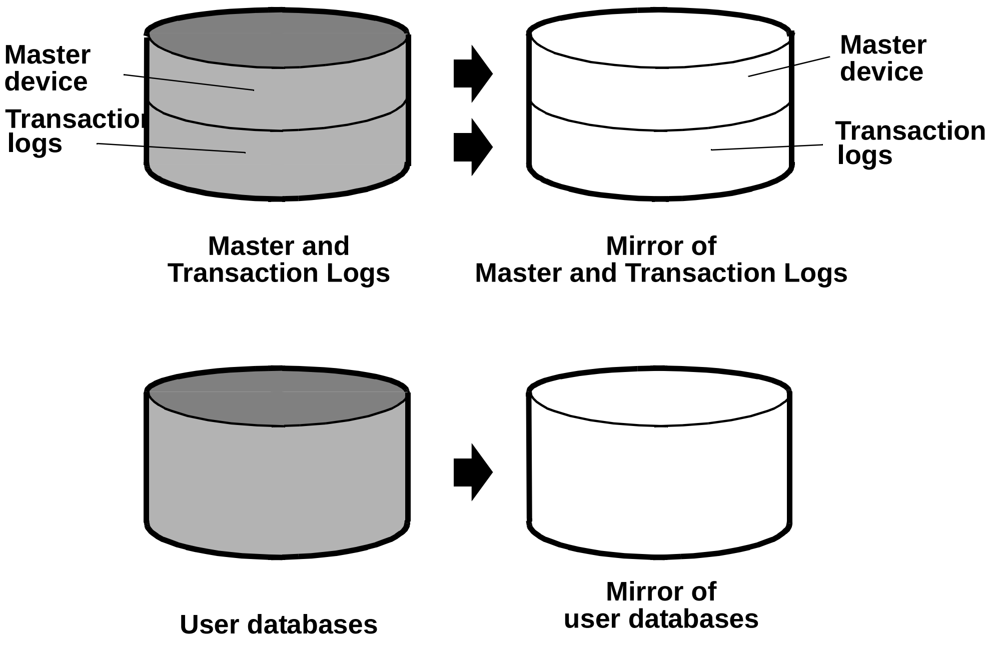
Figure 17-2: Disk mirroring for rapid recovery



With this configuration, all data is written twice, once to the primary disk and once to the mirror. Applications that involve many writes may be slower with disk mirroring than without mirroring.

Figure 17-3 illustrates another configuration with a high level of redundancy. In this configuration, all three database devices are mirrored, but the configuration uses four disks instead of two. This configuration speeds performance during write transactions because the database transaction log is stored on a different device from the user databases, and the system can access both with less disk head travel.

Figure 17-3: Disk mirroring: keeping transaction logs on a separate disk



Conditions that do not disable mirroring

Adaptive Server disables a mirror only when it encounters an I/O error on a mirrored device. For example, if Adaptive Server tries to write to a bad block on the disk, the resulting error disables mirroring for the device. However, processing continues without interruption on the unaffected mirror.

The following conditions *do not* disable a mirror:

- An unused block on a device is bad. Adaptive Server does not detect an I/O error and disables mirroring until it accesses the bad block.

- Data on a device is overwritten. This might happen if a mirrored device is mounted as a UNIX file system, and UNIX overwrites the Adaptive Server data. This causes database corruption, but mirroring is not disabled, since Adaptive Server would not encounter an I/O error.
- Incorrect data is written to both the primary and secondary devices.
- The file permissions on an active device are changed. Some System Administrators may try to test disk mirroring by changing permissions on one device, hoping to trigger I/O failure and unmirror the other device. But the UNIX operating system does not check permissions on a device after opening it, so the I/O failure does not occur until the next time the device is started.

Disk mirroring is not designed to detect or prevent database corruption. Some of the scenarios described can cause corruption, so you should regularly run consistency checks such as dbcc checkalloc and dbcc checkdb on all databases. See Chapter 25, “Checking Database Consistency,” for a discussion of these commands.

Disk mirroring commands

The disk mirror, disk unmirror, and disk remirror commands control disk mirroring. All the commands can be issued while the devices are in use, so you can start or stop database device mirroring while databases are being used.

Note The disk mirror, disk unmirror, and disk remirror commands alter the sysdevices table in the master database. After issuing any of these commands, you should dump the master database to ensure recovery in case master is damaged.

Initializing mirrors

disk mirror starts disk mirroring. *Do not* initialize the mirror device with disk init. A database device and its mirror constitute one logical device. The disk mirror command adds the mirror name to the mirrorname column in the sysdevices table.

Note To retain use of asynchronous I/O, always mirror devices that are capable of asynchronous I/O to other devices capable of asynchronous I/O. In most cases, this means mirroring raw devices to raw devices and operating system files to operating system files.
If the operating system cannot perform asynchronous I/O on files, mirroring a raw device to a regular file produces an error message. Mirroring a regular file to a raw device will work, but will not use asynchronous I/O.

Here is the disk mirror syntax:

```
disk mirror
    name = "device_name" ,
    mirror = "physicalname"
    [ , writes = { serial | noserial }]
```

The *device_name* is the name of the device that you want to mirror, as it is recorded in sysdevices.name (by disk init). Use the mirror = "*physicalname*" clause to specify the path to the mirror device, enclosed in single or double quotes. If the mirror device is a file, "*physicalname*" must unambiguously identify the path where Adaptive Server will create the file; it cannot specify the name of an existing file.

On systems that support asynchronous I/O, the writes option allows you to specify whether writes to the first device must finish before writes to the second device begin (serial) or whether both I/O requests are to be queued immediately, one to each side of the mirror (noserial). In either case, if a write cannot be completed, the I/O error causes the bad device to become unmirrored.

serial writes are the default. The writes to the devices take place consecutively, that is, the first one finishes before the second one starts. serial writes provide protection in the case of power failures: one write may be garbled, but both of them will not be. serial writes are generally slower than noserial writes.

In the following example, tranlog is the logical device name for a raw device. The tranlog device was initialized with disk init and is being used as a transaction log device (as in create database...log on *tranlog*). The following command mirrors the transaction log device:

```
disk mirror
  name = "tranlog",
  mirror = "/dev/rxy1e"
```

Unmirroring a device

Disk mirroring is automatically deactivated when one of the two physical devices fails. When a read or write to a mirrored device is unsuccessful, Adaptive Server prints error messages. Adaptive Server continues to run, unmirrored. You must remirror the disk to restart mirroring.

Use the disk unmirror command to stop the mirroring process during hardware maintenance:

```
disk unmirror
   name = "device_name"
   [, side = { "primary" | secondary }]
   [, mode = { retain | remove }]
```

The side option to the disk unmirror command allows you to specify which side of the mirror to disable. primary (in quotes) is the device listed in the name column of sysdevices; secondary (no quotes required) is the device listed in the mirrorname column of sysdevices. secondary is the default.

The mode option indicates whether the unmirroring process should be temporary (retain) or permanent (remove). retain is the default.

Temporarily deactivating a device

By default (mode=retain), Adaptive Server temporarily deactivates the specified device; you can reactivate it later. This is similar to what happens when a device fails and Adaptive Server activates its mirror:

- I/O is directed only at the remaining device of the mirrored pair.
- The status column of sysdevices is altered to indicate that the mirroring feature has been deactivated.
- The entries for primary (phyname) and secondary (mirrorname) disks are unchanged.

Permanently disabling a mirror

Use mode=remove to disable disk mirroring. This option eliminates all references in the system tables to a mirror device, but does *not* remove an operating system file that has been used as a mirror.

If you set mode=remove:

- The status column is altered to indicate that the mirroring feature is to be ignored.
- The phyname column is replaced by the name of the secondary device in the mirrorname column if the primary device is the one being deactivated.
- The mirrorname column is set to NULL.

Effects on system tables

The mode option changes the status column in sysdevices to indicate that mirroring has been disabled (see Table 16-2 on page 548). Its effects on the phyname and mirrorname columns in sysdevices depend on the side argument also, as shown in Table 17-1

Table 17-1: Effects of mode and side options to the disk mirror command

		side	
		primary	secondary
mode	remove	Name in mirrorname moved to phyname and mirrorname set to null; status changed	Name in mirrorname removed; status changed
	retain	Names unchanged; status changed to indicate which device is being deactivated	

This example suspends the operation of the primary device:

```
disk unmirror
  name = "tranlog",
  side = "primary"
```

Restarting mirrors

Use disk remirror to restart a mirror process that has been suspended due to a device failure or with disk unmirror. The syntax is:

```
disk remirror
    name = "device_name"
```

This command copies the database device to its mirror.

waitfor mirrorexit

Since disk failure can impair system security, you can include the waitfor mirrorexit command in an application to perform specific tasks when a disk becomes unmirrored:

```
begin
   waitfor mirrorexit
     commands to be executed
end
```

The commands depend on your applications. You may want to add certain warnings in applications that perform updates or use sp_dboption to make certain databases read-only if the disk becomes unmirrored.

Note Adaptive Server knows that a device has become unmirrored only when it attempts I/O to the mirror device. On mirrored databases, this occurs at a checkpoint or when the Adaptive Server buffer must be written to disk. On mirrored logs, I/O occurs when a process writes to the log, including any committed transaction that performs data modification, a checkpoint, or a database dump.

waitfor mirrorexit and the error messages that are printed to the console and error log on mirror failure are activated only by these events.

Mirroring the master device

If you choose to mirror the device that contains the master database, in a UNIX environment, you need to edit the runserver file for your Adaptive Server so that the mirror device starts when the server boots.

On UNIX, add the -r flag and the name of the mirror device:

```
dataserver -d /dev/rsd1f -r /dev/rs0e -e/sybase/install/errorlog
```

For information about mirroring the master device on Windows NT, see the *Utility Guide*.

Getting information about devices and mirrors

For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute sp_helpdevice.

Disk mirroring tutorial

The following steps illustrate the use of disk mirroring commands and their effect on selected columns of master..sysdevices. The status number and its hexidecimal equivalent for each entry in sysdevices are in parathesis:

Step 1

Initialize a new test device using:

```
disk init name = "test",
physname = "/usr/sybase/test.dat",
size=5120
```

This inserts the following values into columns of master..sysdevices:

```
name  phyname           mirrorname                                        status
test  /usr/sybase/test.dat   NULL                                          16386
```

Status 16386 indicates that the device is a physical device (2, 0x00000002), and any writes are to a UNIX file (16384, 0x00004000). Since the mirrorname column is null, mirroring is not enabled on this device.

Step 2

Mirror the test device using:

```
disk mirror name = "test",
mirror = "/usr/sybase/test.mir"
```

This changes the master..sysdevices columns to:

```
name  phyname               mirrorname            status
test  /usr/sybase/test.dat  /usr/sybase/test.mir 17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

Step 3

Disable the mirror device (the secondary side), but retain that mirror:

```
disk unmirror name = "test",
side = secondary, mode = retain
```

```
name  phyname               mirrorname           status
test  /usr/sybase/test.dat  /usr/sybase/test.mir  18658
```

Status 18658 indicates that the device is mirrored (64, 0x00000040), and the mirror device has been retained (2048, 0x00000800), but mirroring has been disabled (512 bit off), and only the primary device is used (256 bit off). Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file (16384, 0x00004000) and are in serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

Step 4

Remirror the test device:

```
disk remirror name = "test"
```

This resets the master..sysdevices columns to:

```
name  phyname               mirrorname           status
test  /usr/sybase/test.dat  /usr/sybase/test.mir 17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

Step 5

Disable the test device (the primary side), but retain that mirror:

```
disk unmirror name = "test",
side = "primary", mode = retain
```

This changes the master..sysdevices columns to:

```
name  phyname               mirrorname          status
test  /usr/sybase/test.dat  /usr/sybase/test.mir 16866
```

Status 16866 indicates that the device is mirrored (64, 0x00000040), but mirroring has been disabled (512 bit off) and that only the secondary device is used (256, 0x00000100). Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file (16384, 0x00004000), and are in serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

Step 6

Remirror the test device:

```
disk remirror name = "test"
```

This resets the master..sysdevices columns to:

```
name  phyname               mirrorname           status
test  /usr/sybase/test.dat  /usr/sybase/test.mir 17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

Step 7

Disable the test device (the primary side), and remove that mirror:

```
disk unmirror name = "test", side = "primary",
mode = remove
```

This changes the master..sysdevices columns to:

```
name  phyname              mirrorname
status
test  /usr/sybase/test.mir  NULL                                            16386
```

Status 16386 indicates that the device is a physical device (2, 0x00000002), and any writes are to a UNIX file (16384, 0x00004000). Since the mirrorname column is null, mirroring is not enabled on this device.

Step 8

Remove the test device to complete the tutorial:

```
sp_dropdevice test
```

This removes all entries for the test device from master..sysdevices.

CHAPTER 18 Configuring Memory

This chapter describes how Adaptive Server uses memory and explains how to maximize the memory available to Adaptive Server on your system.

Topics covered in this chapter include:

Determining memory availability for Adaptive Server

The more memory that is available, the more resources Adaptive Server has for internal buffers and caches. Having enough memory available for caches reduces the number of times Adaptive Server has to read data or procedure plans from disk.

There is no performance penalty for configuring Adaptive Server to use the maximum amount of memory available on your computer. However, be sure to assess the other memory needs on your system first, and then configure the Adaptive Server to use only the remaining memory that is still available. Adaptive Server may not be able to start if it cannot acquire the memory for which it is configured.

To determine the maximum amount of memory available on your system for Adaptive Server:

1. Determine the total amount of physical memory on your computer system.
2. Subtract the memory required for the operating system from the total physical memory.

3 Subtract the memory required for Backup Server, Monitor Server, or other Adaptive Server-related software that must run on the same machine.

4 If the machine is not dedicated to Adaptive Server, also subtract the memory requirements for other system uses.

   For example, subtract the memory that will be used by any client applications that will run on the Adaptive Server machine. Windowing systems, such as X Windows, require a lot of memory and can interfere with Adaptive Server performance when used on the same machine as Adaptive Server.

The memory left over after subtracting requirements for the operating system and other applications is the total memory available for Adaptive Server. The value of the max memory configuration parameter specifies the maximum amount of memory to which Adaptive Server is configurable. See "Configuration parameters that affect memory allocation" on page 572 for information about Configure Adaptive Server to use this memory.

How Adaptive Server allocates memory

All database object pages are sized in terms of the **logical page size**, which you specify when you build a new master device. All databases – and all objects in every database– use the same logical page size. The size of Adaptive Server's logical pages (2, 4, 8, or 16K) determines the server's space allocation. Each allocation page, object allocation map (OAM) page, data page, index page, text page, and so on are built on a logical page. For example, if the logical page size of Adaptive Server is 8K, each of these page types are 8K in size. All of these pages consume the entire size specified by the size of the logical page. Larger logical pages allow you to create larger rows, which can improve your performance because Adaptive Server accesses more data each time it reads a page. For example, a 16K page can hold 8 times the amount of data as a 2K page, an 8K page holds 4 times as much data as a 2K page, and so on, for all the sizes for logical pages.

The logical page size is a server-wide setting; you cannot have databases with varying size logical pages within the same server. All tables are appropriately sized so that the row size is no greater than the current page size of the server. That is, rows cannot span multiple pages.

Regardless of the logical page size it is configured for, Adaptive Server allocates space for objects (tables, indexes, text page chains) in extents, each of which is eight logical pages. That is, if a server is configured for 2K logical pages, it allocates one extent, 16K, for each of these objects; if a server is configured for 16K logical pages, it allocates one extent, 128K, for each of these objects.

This is also true for system tables. If your server has many small tables, space consumption can be quite large if the server uses larger logical pages. For example, for a server configured for 2K logical pages, systypes—with approximately 31 short rows, a clustered and a non-clustered index—reserves 3 extents, or 48K of memory. If you migrate the server to use 8K pages, the space reserved for systypes is still 3 extents, 192K of memory. For a server configured for 16K, systypes requires 384K of disk space. For small tables, the space unused in the last extent can become significant on servers using larger logical page sizes.

Databases are also affected by larger page sizes. Each database includes the system catalogs and their indexes. If you migrate from a smaller to larger logical page size, you must account for the amount of disk space each database requires. Table 18-1 lists the minimum size for a database on each of the logical page sizes.

Table 18-1: Minimum database sizes

Logical page size	**Minimum database size**
2K	2MB
4K	4MB
8K	8MB
16K	16MB

Note that the logical page size is not the same as the memory allocation page size. Memory allocation page size is always 2K, regardless of logical pagesize, which can be 2, 4, 8, or 16K. Most memory related configure parameters use units of 2K for their memory pagesize. These configuration parameter include:

- max memory
- total logical memory
- total physical memory
- procedure cache size
- size of process object heap

- size of shared class heap
- size of global fixed heap

Disk space allocation

Note that the logical page size is not the same as the memory allocation page size. This is the unit in which disk space is allocted, and Adaptive Server allocted this space in 2K pages. Some of the configuration parameters use this 2K page size for their allcation units.

Larger logical page sizes and buffers

Adaptive Server allocates buffer pools in units of logical pages. For example, on a server using 2K logical pages, 8MB are allocated to the default data cache. This constitutes approximately 2048 buffers. If you allocated the same 8MB for the default data cache on a server using a 16K logical page size, the default data cache is approximately 256 buffers. On a busy system, this small number of buffers might result in a buffer always being in the wash region, causing a slowdown for tasks requesting clean buffers. In general, to obtain the same buffer management characteristics on larger page sizes as with 2K logical page sizes, you should scale the size of the caches to the larger page size. So, if you increase your logical page size by four times, your cache and pool sizes should be about four times larger as well.

Adaptive Server typically allocates memory dynamically and allocates memory for row processing as it needs it, allocating the maximum size for these buffers, even if large buffers are unnecessary. These memory management requests may cause Adaptive Server to have a marginal loss in performance when handling wide-character data.

Heap memory

A heap memory pool is an internal memory pool created at startup that tasks use to dynamically allocate memory as needed. This memory pool is used by tasks that requirs a lot of memory from the stack, such as tasks that use wide columns. For example, if you make a wide column or row change, the temporary buffer this task uses can be as larger as 16K, which is too big to allocate from the stack. Adaptive Server dynamically allocates and frees memory during the task's run time. The heap memory pool dramatically reduces the pre-declared stack size for each task while also improving the efficiency of memory usage in the server. The heap memory the task uses is returned to the heap memory pool when the task is finished.

Set the heap memory with the heap memory per user configuration parameter.

The size of the memory pool depends on the number of user connections. Sybase recommends that you set heap memory per user to three times the size of your logical page.

How Adaptive Server uses memory

Memory exists in Adaptive Server as total logical or physical memory:

- Total logical memory – is the sum of the memory required for all the sp_configure parameters. The total logical memory is required to be available, but may or may not be in use at a given moment. The total logical memory value may change due to changes in the configuration parameter values.
- Total physical memory – is the sum of all shared memory segments in Adaptive Server. That is, total physical memory is the amount of memory Adaptive Server uses at a given moment. You can verify this value with the read-only configuration parameter total physical memory. The value of total physical memory can only increase because Adaptive Server does not shrink memory pools once they are allocated. You can decrease the amount of total physical memory by changing the configuration parameters and restarting Adaptive Server.

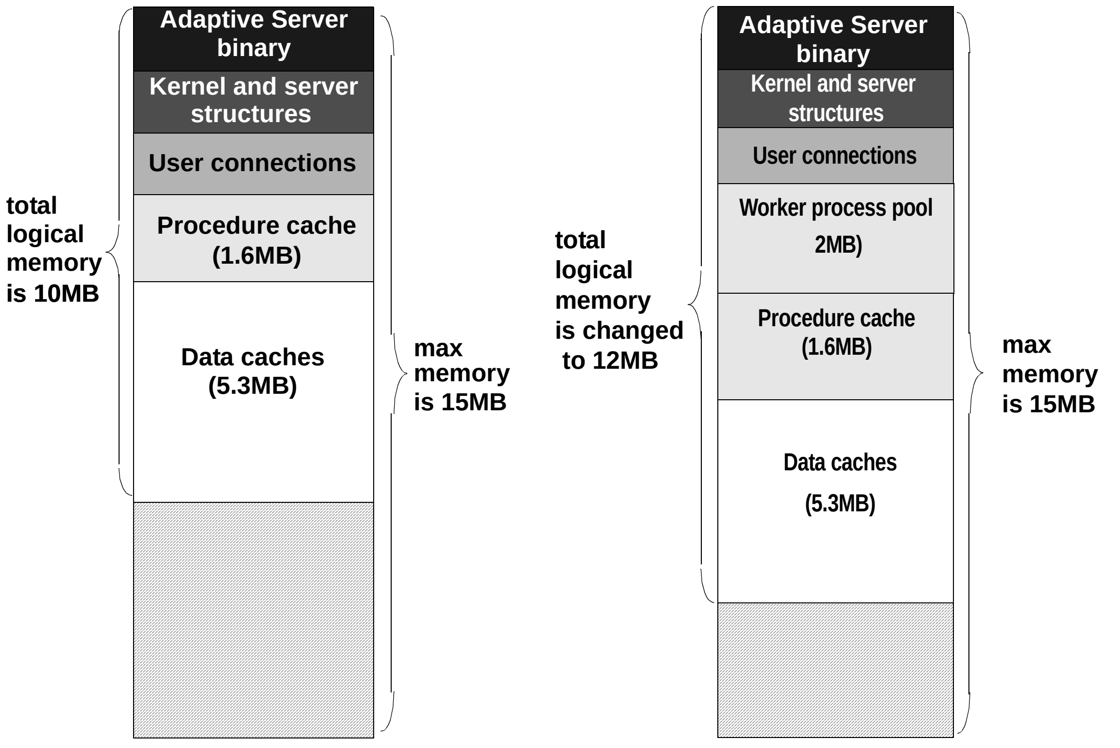
When Adaptive Server starts, it allocates memory for:

- Memory used by Adaptive Server for nonconfigurable data structures

- Memory for all user-configurable parameters, including the data cache, the procedure cache, and the default data cache.

Figure 18-1 illustrates how Adaptive Server allocates memory as you change some of the memory configuration parameters:

Figure 18-1: How Adaptive Server handles memory configuration changes



When a 2MB worker process pool is added to the Adaptive Server memory configuration, the procedure and data caches maintain their originally configured sizes; 1.6MB and 5.3MB, respectively. Because max memory is 5MB larger than the total logical memory size, it easily absorbs the added memory pool. If the new worker process pool brings the size of the server above the limit of max memory, any command you issue to increase the worker process pool fails. If this happens, the total logical memory required for the new configuration is indicated in the sp_configure failure message. Set the value of max memory to a value greater than the total logical memory required by the new configuration. Then retry your sp_configure request.

The size of the default data cache and the procedure cache has a significant impact on overall performance. See Chapter 14, “Memory Use and Performance,” in the *Performance and Tuning Guide* for recommendations on optimizing procedure cache size.

How much memory does Adaptive Server need?

The total memory Adaptive Server requires to start is the *sum of all memory configuration parameters* plus the *size of the procedure cache* plus the *size of the buffer cache*, where the size of the procedure cache and the size of the buffer cache are expressed in round numbers rather than in percentages. The procedure cache size and buffer cache size do *not* depend on the total memory you configure. You can configure the procedure cache size and buffer cache size independently. Use sp_cacheconfig to obtain information such as the total size of each cache, the number of pools for each cache, the size of each pool, and so on.

Use sp_configure to determine the total amount of memory Adaptive Server is using at a given moment. For example:

```
1> sp_configure "total logical memory"
Parameter Name                 Default     Memory Used Config Value Run Value
Unit                 Type
------------------------------ ----------- ----------- ------------ ---------
-------------------- ----------
total logical memory                 33792      127550        63775       63775
memory pages(2k)     read-only
```

The value for the Memory Used column is represented in kilobytes, while the value for the Config Value column is represented in 2K pages.

The config value indicates the total logical memory Adaptive Server uses while it is running. The run value column shows the total logical memory being consumed by the current Adaptive Server configuration. You will see a different output if you run this example because no two Adaptive Servers are likely to be configured in exactly the same fashion.

For more information about sp_configure please see *“Sybase Adaptive Server Enterprise Reference Manual Volume 3: Procedures”*.

If you are upgrading

If you upgrade to release 12.5 Adaptive Server or higher, pre-12.5 Adaptive Server configuration values for total logical memory, procedure cache percent, and min online engines are used to calculate the new values for procedure cache size and number of engines at startup. Adaptive Server computes the size of the default data cache during the upgrade and writes this value to the configuration file. If the computed sizes of the data cache or procedure cache are less than the default sizes, they are reset to the default. During the upgrade, max memory is set to the value of total logical memory specified in the configuration file.

You should reset the value of max memory to comply with the resource requirements.

You can use the verify option of sp_configure to verify any changes you make to the configuration file without having to restart Adaptive Server. The syntax is:

```
sp_configure “configuration file”, 0, “verify”, “full_path_to_file”
```

Configuration parameters that affect memory allocation

When setting Adaptive Server’s memory configuration, you specify each memory requirement with an absolute value, using sp_configure. You also specify the size of the procedure and default data caches in an absolute value.

There are three configuration parameters that affect the way in which memory is allocated. They are:

max memory

The configuration parameter max memory allows you to establish a maximum setting for the amount of memory you can allocate to Adaptive Server. Setting max memory to a slightly larger value than immediately necessary provides extra memory that is utilized when Adaptive Server’s memory needs are increased.

The way Adaptive Server allocates the memory specified by max memory depends on how you configure allocate max shared memory and dynamic allocation on demand.

allocate max shared memory

The allocate max shared memory parameter allows you to either allocate all the memory specified by max memory at start-up or to allocate only the memory required by the total logical memory specification during start-up.

On some platforms, if the number of shared memory segments allocated to an application is more than an optimal, platform-specific number, it could result in performance degradation. If this occurs, set max memory to the maximum amoun available for ASE. Set allocate max shared memory to one and reboot the server. This ensures that all the memory for max memory will be allocated by ASE during boot time with the least number of segments.

For example, if you set allocate max shared memory to 0 (the default) and max memory to 500MB, but the server configuration only requires 100MB of memory at start-up, Adaptive Server allocates the remaining 400MB only when it requires the additional memory. However, if you set allocate max shared memory to 1, Adaptive Server allocates the entire 500MB when it starts.

The advantage of allocating all the memory at start-up, by setting allocate max shared memory to 1, is that there is no performance degradation while the server is readjusting for the additional memory. However, if you do not predict memory growth properly, and max memory is set to a large value, you may be wasting physical memory. Since you cannot dynamically decrease memory configuration parameters, it is important that you take into account other memory requirements.

dynamic allocation on demand

dynamic allocation on demand allows you to determine whether your memory resources are allocated as soon as they are requested or only as they are needed. Setting dynamic allocation on demand to 1 allocates memory changes as needed, and setting it to 0 allocates the configured memory requested in the memory configuration change at the time of the memory reconfiguration.

For example, if you set the value of dynamic allocation on demand to 1 and you change number of user connections to 1024, the total logical memory is 1024 multiplied by the amount of memory per user. If the amount of memory per user is 112K, then the memory for user connections is 112MB (1024 x 112).

This is the maximum amount of memory that the number of user connections configuration parameter is allowed to use. However, if only 500 users are connected to the server, the amount of total physical memory used by the number of user connections parameter is 56MB (500 x 112).

Now assume the value of dynamic allocation on demand is 0; when you change number of user connections to 1024, all user connection resources are configured immediately.

Optimally, you should organize Adaptive Server's memory so that the amount of total physical memory is less than the amount of total logical memory, which is less than the max memory. This can be achieved, in part, by setting the value of dynamic allocation on demand to 1, and setting the value of allocate max shared memory to 0.

Dynamically allocating memory

Adaptive Server allocates physical memory dynamically. This allows users to change the memory configuration of Adaptive Server without restarting the server.

Note Adaptive Server allocates memory dynamically; however, it does not decrease memory dynamically. It is important that you accurately assess the needs of your system, because you may need to restart the server if you decrease the memory configuration parameters and want to release previously used physical memory. See "Dynamically decreasing memory configuration parameters" on page 575 for more information.

Consider changing the value of the max_memory configuration parameter:

- When you change the amount of RAM on your machine
- When the pattern of use of your machine changes
- When the configuration fails because max_memory is insufficient.

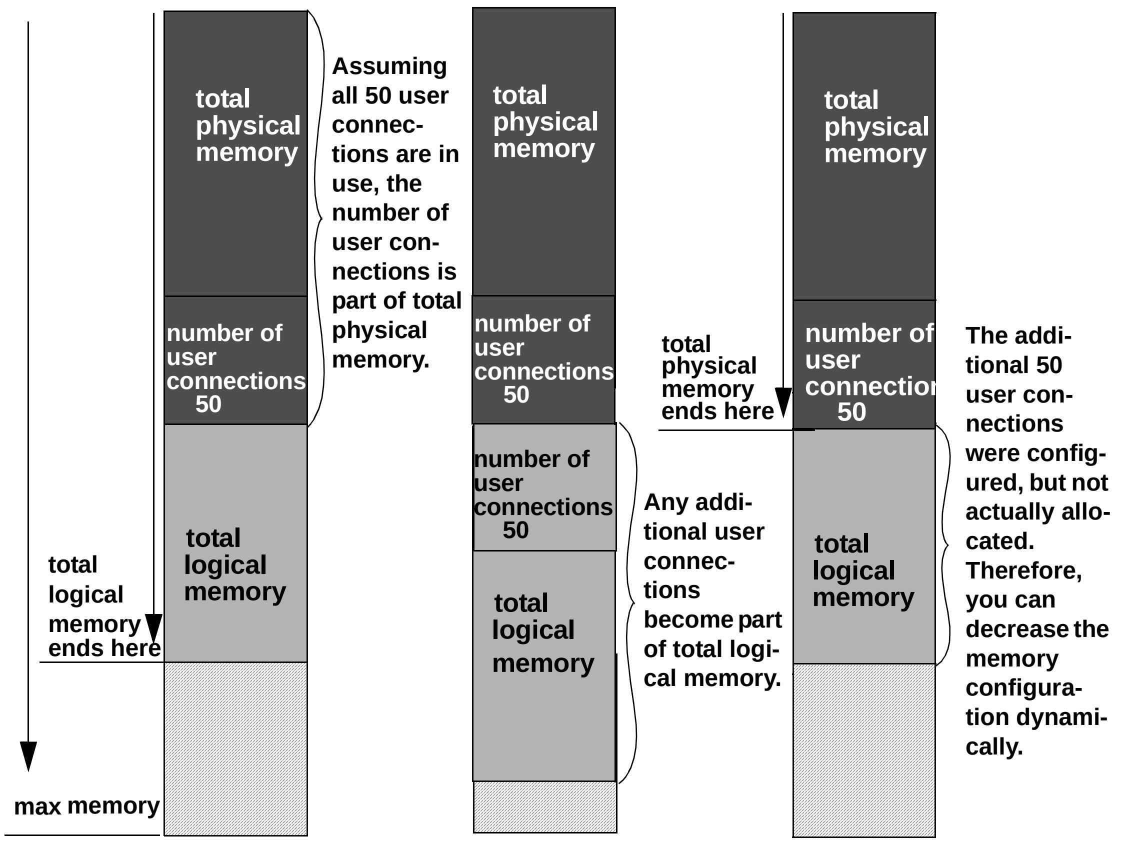
If Adaptive Server cannot start

When Adaptive Server starts, it must acquire the full amount of memory set by total logical memory from the operating system. If Adaptive Server cannot start because it cannot acquire enough memory, reduce the memory requirements by reducing the values for the configuration parameters that consume memory. You may also need to reduce the values for other configuration parameters that require large amounts of memory. Then restart Adaptive Server to use the new values. See Chapter 5, “Setting Configuration Parameters,” for information about using configuration files.

Dynamically decreasing memory configuration parameters

If you reset memory configuration parameters to a lower value, any engaged memory will not be released dynamically. To see how the changes in memory configuration are decreased, see Figure 18-2 and Figure 18-3.

Figure 18-2: dynamic allocation on demand set to 1 with no new user connections


total physical memory
number of user connections 50
total logical memory
total logical memory ends here
max memory
Assuming all 50 user connections are in use, the number of user connections is part of total physical memory.
total physical memory
number of user connections 50
number of user connections 50
total logical memory
Any additional user connections become part of total logical memory.
total physical memory ends here
total physical memory
number of user connections 50
total logical memory
The additional 50 user connections were configured, but not actually allocated. Therefore, you can decrease the memory configuration dynamically.

In Figure 18-2, because dynamic allocation on demand is set to 1, memory is now used only when there is an event that triggers a need for additional memory use. In this example, such an event would be a request for additional user connections, when a client attempts to log into Adaptive Server.

You may decrease number of user connections to a number that is greater than or equal to the number of user connetions actually allocated, because, with dynamic allocation on demand set to 1, and without an actual increase in user connection request, no additional memory is required from the server.

Figure 18-3: dynamic allocation on demand set to 1, with new user connections logged on

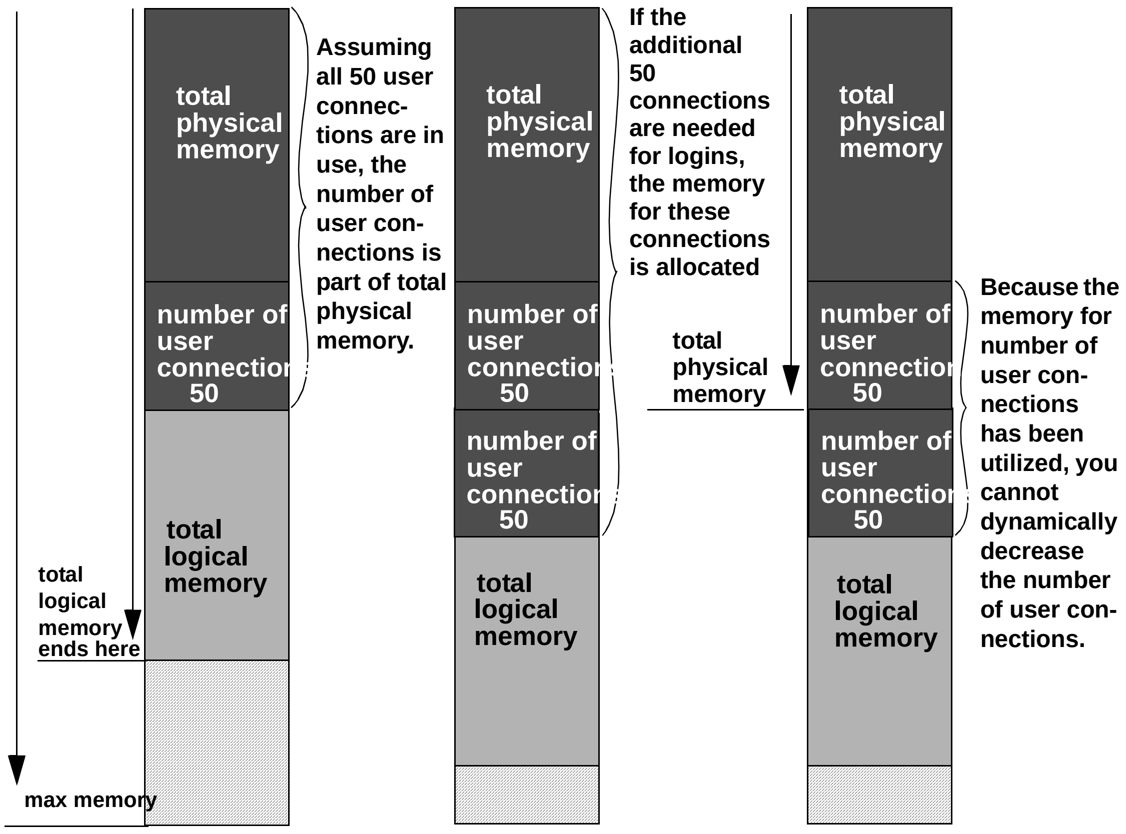
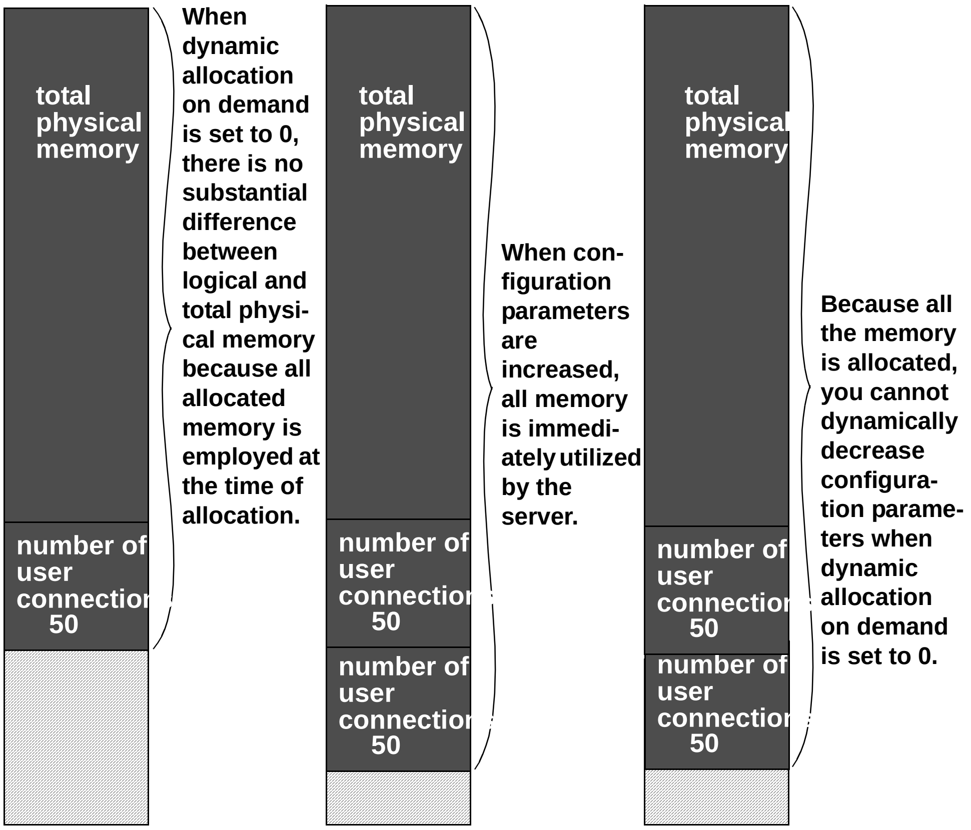

total physical memory
number of user connections 50
total logical memory
total logical memory ends here
max memory
Assuming all 50 user connec-tions are in use, the number of user con-nections is part of total physical memory.
total physical memory
number of user connections 50
number of user connections 50
total logical memory
If the additional 50 connections are needed for logins, the memory for these connections is allocated
total physical memory
total physical memory
number of user connections 50
number of user connections 50
total logical memory
Because the memory for number of user con-nections has been utilized, you cannot dynamically decrease the number of user con-nections.

Figure 18-3 assumes that each of the additional 50 user connections is actually used. You cannot decrease number of user connections, because the memory is in use. You can use sp_configure to specify a change to memory configuration parameters, but this change will not take place until the server is restarted.

Figure 18-4: dynamic allocation on demand set to 0


total physical memory
number of user connections 50
When dynamic allocation on demand is set to 0, there is no substantial difference between logical and total physical memory because all allocated memory is employed at the time of allocation.
total physical memory
number of user connections 50
number of user connections 50
When configuration parameters are increased, all memory is immediately utilized by the server.
total physical memory
number of user connections 50
number of user connections 50
Because all the memory is allocated, you cannot dynamically decrease configuration parameters when dynamic allocation on demand is set to 0.

Note In theory, when dynamic allocation on demand is set to 0, there should be no difference between total logical and physical memory. However, there are some discrepancies in the way that Adaptive Server estimates memory needs, and the way in which memory is actually required for usage. For this reason, you may see a difference between the two during runtime.

When dynamic allocation on demand is set to 0, all configured memory requirements are immediately allocated. You cannot dynamically decrease memory configuration.

In Figure 18-3 and Figure 18-4, users can change the memory configuration parameter values to any smaller, valid value. While this change does not take place dynamically, it disallows new memory use. For example, if you have configured number of user connections to allow for 100 user connections and then change that value to 50 user connections, in the situations represented by Figure 18-3 and Figure 18-4 you can decrease the number of user connections value back to 50. This change does not effect the memory used by Adaptive Server until after the server is restarted, but it prevents any new users from logging onto the server.

System procedures for configuring memory

The three system procedures that you need to use while configuring Adaptive Server memory are:

- sp_configure
- sp_helpconfig
- sp_monitorconfig

Using *sp_configure* to set configuration parameters

The full syntax and usage of sp_configure and details on each configuration parameter are covered in Chapter 5, “Setting Configuration Parameters.” The rest of this chapter discusses issues pertinent to configuring the parameters that use Adaptive Server memory.

Execute sp_configure, specifying “Memory Use,” to see these parameter settings on your server.

```
sp_configure "Memory Use"
```

A “#” in the “Memory Used” column indicates that this parameter is a component of another parameter and that its memory use is included in the memory use for the other component. For example, memory used for stack size and stack guard size contributes to the memory requirements for each user connection and worker process, so the value is included in the memory required for number of user connections and for number of worker processes, if this is more than 200.

Some of the values in this list are computed values. They cannot be set directly with sp_configure, but are reported to show where memory is allocated. Among the computed values is total data cache size.

How much memory is available for dynamic growth?

Issueing sp_configure memory displays all of the memory parameters and determines the difference between max memory and total logical memory, which is the amount of memory available for dynamic growth. For example:

```
1> sp_configure memory
Msg 17411, Level 16, State 1:
Procedure ’sp_configure’, Line 187:
Configuration option is not unique.
Parameter Name                 Default     Memory Used Config Value Run Value
Unit                 Type
------------------------------ ----------- ----------- ------------ ----------
-------------------- ----------
additional network memory                0           0            0            0
bytes                dynamic
allocate max shared memory               0           0            0            0
switch               dynamic
heap memory per user                  4096           0         4096         4096
bytes                dynamic
lock shared memory                       0           0            0            0
switch               static
max memory                           33792      300000       150000       150000
memory pages(2k)     dynamic
memory alignment boundary            16384           0        16384        16384
bytes                static
memory per worker process             1024           4         1024         1024
bytes                dynamic
shared memory starting address           0           0            0            0
not applicable       static
total logical memory                 33792      110994        55497        55497
memory pages(2k)     read-only
total physical memory                    0       97656            0        48828
memory pages(2k)     read-only

An additional 189006 K bytes of memory is available for reconfiguration. This is
the difference between ’max memory’ and ’total logical memory’.
```

Using *sp_helpconfig* to get help on configuration parameters

sp_helpconfig estimates the amount of memory required for a given configuration parameter and value. It also provides a short description of the parameter, information about the minimum, maximum, and default values for the parameter, the run value, and the amount of memory used at the current run value. sp_helpconfig is particularly useful if you are planning substantial changes to a server, such as loading large, existing databases from other servers, and you want to estimate how much memory is needed.

To see how much memory is required to configure a parameter, enter enough of the parameter name so that it is a unique name and the value you want to configure:

```
sp_helpconfig "worker processes", "50"

number of worker processes is the maximum number of worker processes that can be
in use Server-wide at any one time.

Minimum Value Maximum Value   Default Value    Current Value       Memory Used
Unit          Type
------------- -------------- -------------    -------------       ----------
------------ ------------
0             2147483647      0                0                   0

number        dynamic

Configuration parameter, 'number of worker processes', will consume 7091K of
memory if configured at 50.
Changing the value of 'number of worker processes' to '50' increases the amount
of memory ASE uses by 7178 K.
```

You can also use sp_helpconfig to determine the value to use for sp_configure, if you know how much memory you want to allocate to a specific resource:

```
sp_helpconfig "user connections", "5M"
number of user connections sets the maximum number of user connections that can
be connected to SQL Server at one time.
Minimum Value  Maximum Value   Default Value    Current Value    Memory Used
Unit                    Type

----------    -----------     -------------    -------------    -----------
-------------------- ----------
5             2147483647       25               25              3773
number                 dynamic
Configuration parameter, 'number of user connections', can be configured to 33
to fit in 5M of memory.
```

The important difference between the syntax of these two statements is the use of a unit of measure in the second example to indicate to the procedure that the value is a size, not a configuration value. The valid units of measure are:

- P – pages, (Adaptive Server 2K pages)
- K – kilobytes
- M – megabytes
- G – gigabytes

There are some cases where the syntax does not make sense for the type of parameter, or where Adaptive Server is not able to calculate the memory use. sp_helpconfig prints an error message in these cases. For example if you attempt to specify a size for a parameter that toggles, such as allow resource limits, sp_helpconfig prints the message that describes the function of the parameter for all the configuration parameters that do not use memory.

Using *sp_monitorconfig* to find metadata cache usage statistics

sp_monitorconfig displays metadata cache usage statistics on certain shared server resources, including:

- The number of databases, objects, and indexes that can be open at any one time
- The number of auxiliary scan descriptors used by referential integrity queries
- The number of free and active descriptors
- The percentage of active descriptors
- The maximum number of descriptors used since the server was last started
- The current size of the procedure cache and the amount actually used.

For example, suppose you have configured the number of open indexes configuration parameter to 500. During a peak period, you can run sp_monitorconfig as follows to get an accurate reading of the actual metadata cache usage for index descriptors. For example:

```
sp_monitorconfig "number of open indexes"
Usage information at date and time: Aug 14 1997 8:54AM.
```

```
Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
number of open  217       283      56.60      300               No
objects
```

In this report, the maximum number of open indexes used since the server was last started is 300, even though Adaptive Server is configured for 500. Therefore, you can reset the number of open indexes configuration parameter to 330, to accommodate the 300 maximum used index descriptors, plus space for 10 percent more.

You can also determine the current size of the procedure cache with sp_monitorconfig procedure cache size parameter. This parameter describes the amount of space in the procedure cache is currently configured for and the most it has ever actually used. For example, the procedure cache in the following server is configured for 20,000 pages:

```
1> sp_monitorconfig "procedure cache size"
option_name                    config_value run_value
------------------------------ ------------ -----------
procedure cache size                   3271        3271
```

However, when you run sp_montorconfig "procedure cache size", you find that the most the procedure cache has ever used is 14241 pages, which means that you can lower the run value of the procedure cache, saving memory:

```
1> sp_monitorconfig "procedure cache size"
Usage information at date and time: May  1 2001 10:30AM.
Name                            # Free      # Active    % Active    # Max Ever
Used  Re-used
------------------------------  ----------  ----------  ----------
--------------  -------
procedure cache size            5878        14122       70.61       14241

    No
```

Major uses of Adaptive Server memory

This section discusses configuration parameters that use large amounts of Adaptive Server memory and those that are commonly changed at a large number of Adaptive Server installations. These parameters should be checked by System Administrators who are configuring an Adaptive Server for the first time. System Administrators should review these parameters when the system configuration changes, after upgrading to a new release of Adaptive Server, or when making changes to other configuration variables that use memory.

Configuration parameters that use less memory, or that are less frequently used, are discussed in “Other parameters that use memory” on page 589.

Adaptive Server executable code and overhead

The size of the executable code is included in the value for total logical memory. The size of the executable code plus overhead varies by platform and release, but generally ranges from 6MB to 8MB. To determine the size of the Adaptive Server executable and overhead for your platform, use sp_configure to display the value of the executable codesize + overhead configuration parameter. See “executable codesize + overhead” on page 141 for more information.

When you enable Component Integration Services with the enable cis configuration parameter and then restart Adaptive Server, the size of the executable code and overhead increases. Other Component Integration Services configuration parameters use memory from the general pool of memory.

Data and procedure caches

As explained in “How Adaptive Server uses memory” on page 569, you specify the size of the data and procedure caches. Having sufficient data and procedure cache space is one of the most significant contributors to performance. This section explains the details between the two caches and how to monitor cache sizes.

Determining the procedure cache size

procedure cache size specifies the size of your procedure cache in 2K pages, regardless of the server’s logical page size. For example:

```
          sp_configure “procedure cache size”
1> sp_configure "procedure cache size"
Parameter Name                 Default     Memory Used Config Value
Run Value   Unit                 Type
------------------------------ ----------- ----------- ------------
----------- -------------------- ----------
procedure cache size                  3271        6914        3271
             3271 memory pages(2k) dynamic
```

The amount of memory used for the procedure cache is 8.248MB. To set the procedure cache to a different size, issue the following:

```
sp_configure “procedure cache size”, new_size
```

This example resets the procedure cache size to 10000 2K pages (20MB):

```
sp_configure “procedure cache size”, 10000
```

Determining the default data cache size

Both sp_cacheconfig and sp_helpcache display the current default data cache in megabytes. For example, the following shows an Adaptive Server configured with 19.86MB of default data cache:

```
sp_cacheconfig


Cache Name                Status     Type       Config Value    Run Value
------------------        --------   --------   ------------    ----------
default data cache        Active     Default    0.00 Mb         19.86Mb
                                                ------------    --------
                        Total                   0.00Mb          19.86 Mb
========================================================================
Cache: default data cache, Status: Active, Type: Default
       Config Size: 0.00 Mb, Run Size: 19.86 Mb
       Config Replacement: strict LRU, Run Replacement: strict LRU
       Config Partition:    1,   Run Partition:    1
IO Size       Wash Size     Config Size     Run Size      APF Percent
--------      ---------     ------------    ----------    -----------
2 Kb          4066 Kb        0.00 Mb         19.86 Mb               10
```

To change the default data cache, issue sp_cacheconfig, and specify "default data cache." For example, to change the default data cache to 25MB, enter:

```
sp_cacheconfig “default data cache”, “25M”
```

You must restart Adaptive Server for this change to take effect.

The default data cache size is an absolute value, and the minimum size for the cache size is 256 times the logical pagesize; for 2K, the minimum is 512K, for 16K, the minimum is 4M. The default value is 8MB. During the upgrade process, Adaptive Server sets the default data cache size to the value of the default data cache in the configuration file.

Monitoring cache space

You can check data cache and procedure cache space with sp_configure:

```
sp_configure "total data cache size"
```

Monitoring cache sizes using the errorlog

Another way to determine how Adaptive Server uses memory is to examine the memory-related messages written to the error log when Adaptive Server starts. These messages state exactly how much data and procedure cache is allocated, how many **compiled objects** can reside in cache at any one time, and buffer pool sizes.

These messages provide the most accurate information regarding cache allocation on Adaptive Server. As discussed earlier, the amount of memory allocated for the procedure caches depends on the run value of the procedure cache size configuration parameter.

Each of these error log messages is described below.

Procedure cache messages

Two error log messages provide information about the procedure cache.

```
server: Number of proc buffers allocated: 556
```

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

```
server: Number of blocks left for proc headers: 629
```

This message indicates the total number of procedure headers (proc headers) available for use in the procedure cache.

proc buffer

A *proc buffer* (procedure buffer) is a data structure used to manage compiled objects in the procedure cache. One proc buffer is used for every copy of a compiled object stored in the procedure cache. When Adaptive 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.

proc header

A *proc header* (procedure header) is where a compiled object is stored while in the procedure cache. 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.

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.

Data cache messages

When Adaptive Server starts, it records the total size of each cache and the size of each pool in the cache in the error log. This example shows the default data cache with two pools and a user-defined cache with two pools:

```
Memory allocated for the default data cache cache: 8030 Kb
Size of the 2K memory pool: 7006 Kb
Size of the 16K memory pool: 1024 Kb
Memory allocated for the tuncache cache: 1024 Kb
Size of the 2K memory pool: 512 Kb
Size of the 16K memory pool: 512 Kb
```

User connections

The amount of memory required per user connection varies by platform, and it changes when you change other configuration variables, including:

- default network packet size
- stack size and stack guard size
- user log cache size

Changing any of these parameters changes the amount of space used by each user connection: you have to multiply the difference in size by the number of user connections. For example, if you have 300 user connections, and you are considering increasing the stack size from 34K to 40K, the new value requires 1800K more memory.

Open databases, open indexes, and open objects

The three configuration parameters that control the total number of databases, indexes, and objects that can be open at one time are managed by special caches called **metadata caches**. The metadata caches reside in the kernel and server structures portion of Adaptive Server memory. You configure space for each of these caches with these parameters:

- number of open databases
- number of open indexes
- number of open objects

When Adaptive Server opens a database or accesses an index or an object, it needs to read information about it in the corresponding system tables: sysdatabases, sysindexes, and sysobjects. The metadata caches for databases, indexes, or objects let Adaptive Server access the information that describes it in the sysdatabases, sysindexes, or sysobjects row directly in its in-memory structure. This improves performance because Adaptive Server bypasses expensive calls that require disk access. It also reduces synchronization and spinlock contention when Adaptive Server has to retrieve database, index, or object information at runtime.

Managing individual metadata caches for databases, indexes, or objects is beneficial for a database that contains a large number of indexes and objects and where there is high concurrency among users. For more information about configuring the number of metadata caches, see “number of open databases” on page 142, “number of open indexes” on page 144, and “number of open objects” on page 145.

Number of locks

All processes in Adaptive Server share a pool of lock structures. As a first estimate for configuring the number of locks, multiply the number of concurrent user connections you expect, *plus* the number of worker processes that you have configured, by 20. The number of locks required by queries can vary widely. See “number of locks” on page 134 for more information. For information on how worker processes use memory, see “Worker processes” on page 590.

Database devices and disk I/O structures

The number of devices configuration parameter controls the number of database devices that can be used by Adaptive Server for storing data. See “number of devices” on page 111 for more information.

When a user process needs to perform a physical I/O, the I/O is queued in a disk I/O structure. See “disk i/o structures” on page 110 for more information.

Other parameters that use memory

This section discusses configuration parameters that use moderate amounts of memory or are infrequently used.

Parallel processing

Parallel processing requires more memory than serial processing. The configuration parameters that affect parallel processing are:

- number of worker processes
- memory per worker process
- partition groups
- number of mailboxes and number of messages

Worker processes

The configuration parameter number of worker processes sets the total number of worker processes available at one time in Adaptive Server. Each worker process requires about the same amount of memory as a user connection.

Changing any of these parameters changes the amount of memory required for each worker process:

- default network packet size
- stack size and stack guard size
- user log cache size
- memory per worker process

The memory per worker process configuration parameter controls the additional memory that is placed in a pool for all worker processes. This additional memory stores miscellaneous data structure overhead and inter-worker process communication buffers. See the *Performance and Tuning Guide* for information on setting memory per worker process.

Parallel queries and the procedure cache

Each worker process makes its own copy of the query plan in space borrowed from the procedure cache. The coordinating process keeps two copies of the query plan in memory.

Partition groups

You need to reconfigure the value only if you use a very large number of partitions in the tables on your server. See “partition groups” on page 203 for more information.

Remote servers

Some configuration parameters that allow Adaptive Server to communicate with other Sybase servers such as Backup Server, Component Integration Services, or XP Server use memory.

The configuration parameters that affect remote servers and that use memory are:

- number of remote sites

- number of remote connections
- number of remote logins
- remote server pre-read packets

Number of remote sites

Set the number of remote sites configuration parameter to the number of simultaneous sites you need to communicate to or from on your server. If you use only Backup Server, and no other remote servers, you can increase your data cache and procedure cache space by reducing this parameter to 1.

The connection from Adaptive Server to XP Server uses one remote site.

Other configuration parameters for RPCs

These configuration parameters for remote communication use only a small amount of memory for each connection:

- number of remote connections
- number of remote logins

Each simultaneous connection from Adaptive Server to XP Server for ESP execution uses one remote connection and one remote login.

Since the remote server pre-read packets parameter increases the space required for each connection configured by the number of remote connections parameter, increasing the number of pre-read packets can have a significant effect on memory use.

Referential integrity

If the tables in your databases use a large number of referential constraints, you may need to adjust the number of aux scan descriptors parameter, if user connections exceed the default setting. In most cases, the default setting is sufficient. If a user connection exceeds the current setting, Adaptive Server returns an error message suggesting that you increase the number of aux scan descriptors parameter setting.

Other parameters that affect memory

Other parameters that affect memory are listed below. When you reset these configuration parameters, check the amount of memory they use and the effects of the change on your procedure and data cache.

• additional network memory	• max SQL text monitored
• allow resource limits	• number of alarms
• audit queue size	• number of large i/o buffers
• event buffers per engine	• permission cache entries
• max number network listeners	
• max online engines	

CHAPTER 19 Configuring Data Caches

This chapter describes how to create and administer named caches on Adaptive Server.

The most common reason for administering data caches is to reconfigure them for performance. This chapter is primarily concerned with the mechanics of working with data caches. Chapter 32, “Memory Use and Performance,” in the *Performance and Tuning Guide* discusses performance concepts associated with data caches.

The data cache on Adaptive Server

The data cache holds the data, index, and log pages currently in use as well as pages used recently by Adaptive Server. When you first install Adaptive Server, it has a single default data cache that is used for all data, index, and log activity. The default size of this cache is 8K. Creating other caches does not reduce the size of the default data cache. Also, you can create pools within the named caches and the default cache to perform large I/Os. You can then bind a database, table (including the syslogs table), index, or text or image page chain to a named data cache.

Large I/O sizes enable Adaptive Server to perform data prefetching when the query optimizer determines that prefetching would improve performance. For example, an I/O size of 128K on a server configured with 16K logical pages means that Adaptive Server can read an entire extent —8 pages—all at once, rather than performing 8 separate I/Os. See “Understanding the Query Optimizer,” in the *Performance and Tuning Guide* for details about the optimizer.

Sorts can also take advantage of buffer pools configured for large I/O sizes.

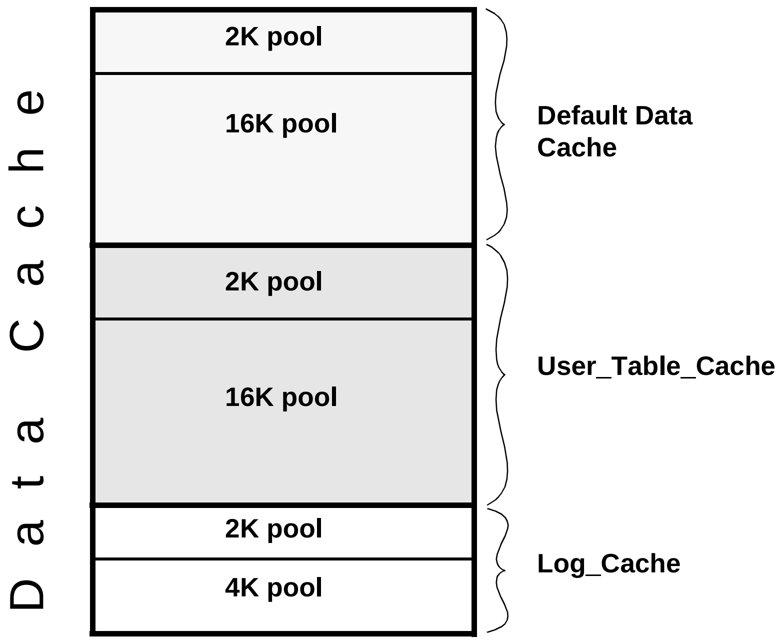
Configuring named data caches does not divide the default cache into separate cache structures. The named data caches that you create can be used only by databases or database objects that are explicitly bound to them. All objects not explicitly bound to named data caches use the default data cache.

Adaptive Server provides user-configurable data caches to improve performance, especially for multiprocessor servers. See “The Data Cache” on page 32-7 of the *Performance and Tuning Guide.*

Figure 19-1 shows a data cache with the default cache and two named data caches. This server uses 2K logical pages.

The default cache contains a 2K pool and a 16K pool. The User_Table_Cache cache also has a 2K pool and a 16K pool. The Log_Cache has a 2K pool and a 4K pool.

Figure 19-1: Data cache with default cache and two named data caches



Cache configuration commands

The following table lists commands for configuring named data caches, for binding and unbinding objects to caches, and for reporting on cache bindings. It also lists procedures you can use to check the size of your database objects, and commands that control cache usage at the object, command, or session level.

Command	Function
sp_cacheconfig	Creates or drops named caches, and changes the size, cache type, cache policy, or number of cache partitions.
sp_poolconfig	Creates and drops I/O pools, and changes their size, wash size, and asynchronous prefetch percent limit.

Command	Function
sp_bindcache	Binds databases or database objects to a cache.
sp_unbindcache	Unbinds specific objects or databases from a cache.
sp_unbindcache_all	Unbinds all objects bound to a specified cache.
sp_helpcache	Reports summary information about data caches and lists the databases and database objects that are bound to caches.
sp_cachestrategy	Reports on cache strategies set for a table or index, and disables or reenables prefetching or MRU strategy.
sp_logiosize	Changes the default I/O size for the log.
sp_spaceused	Provides information about the size of tables and indexes or the amount of space used in a database.
sp_estspace	Estimates the size of tables and indexes, given the number of rows the table will contain.
sp_help	Reports the cache a table is bound to.
sp_helpindex	Reports the cache an index is bound to.
sp_helpdb	Reports the cache a database is bound to.
set showplan on	Reports on I/O size and cache utilization strategies for a query.
set statistics io on	Reports number of reads performed for a query.
set prefetch [on |off]	Enables or disables prefetching for an individual session.
select... (prefetch...lru | mru)	Forces the server to use the specified I/O size or MRU replacement strategy.

In addition to using the commands to configure named data caches interactively, you can also edit the configuration file located in the *$SYBASE* directory. See“Configuring data caches with the configuration file” on page 628 for more information.

Information on data caches

Use sp_cacheconfig to create and configure named data caches. When you first install Adaptive Server, it has a single cache named default data cache. To see information about caches, type:

```
sp_cacheconfig
```

The results of sp_cacheconfig look similar to:

```
Cache Name                 Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ------------
default data cache        Active    Default       0.00 Mb     59.44 Mb
                                             ------------ ------------
                                    Total         0.00 Mb     59.44 Mb

======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 0.00 Mb,   Run Size: 59.44 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb  12174 Kb      0.00 Mb     59.44 Mb     10
```

Summary information for each cache is printed in a block at the top of the report, ending with a total size for all configured caches. For each cache, there is a block of information reporting the configuration for the memory pools in the cache.

The columns are:

- “Cache Name” gives the name of the cache.
- “Status” indicates whether the cache is active. Possible values are:
    - “Pend/Act” – the cache was just created and will be active after a restart.
    - “Active” – the cache is currently active.
    - “Pend/Del” – the cache is active, but will be deleted at the next restart of the server. The cache size was reset to 0 interactively. See “Dropping data caches” on page 621 for more information.
- “Type” indicates whether the cache can store data and log pages (“Mixed”) or log pages only (“Log Only”). Only the default cache has the type “Default.” You cannot change the type of the default data cache or change the type of any other cache to “Default.”
- “Config Value” displays the size of the cache after the next restart of Adaptive Server. In the preceding example output, the default data cache has not been explicitly configured, so its size is 0.
- “Run Value” displays the size that Adaptive Server is currently using. For the default data cache, this size is always the amount of all data cache space that is not explicitly configured to another cache.

The second block of output begins with three lines of information that describe the cache. The first two lines repeat information from the summary block at the top. On the third line, "Config Replacement" and "Run Replacement" show the cache policy, which is either "strict LRU" or "relaxed LRU." The run setting is the setting in effect; if the policy has been changed since the server was restarted, the config setting will be different from the run setting.

sp_cacheconfig then provides a row of information for each pool in the cache:

- "IO Size" shows the size of the buffers in the pool. The default size of the pool is the size of the server's logical page. When you first configure a cache, all the space is assigned to the pool. Valid sizes are 2K, 4K, 8K, and 16K.
- "Wash Size" indicates the wash size for the pool. See"Changing the wash area for a memory pool" on page 614 for more information.
- "Config Size" and "Run Size" display the configured size and the size currently in use. These values differ for the pool because you cannot explicitly configure its size. These may differ for other pools if you have tried to move space between them, and some of the space could not be freed.
- "Config Partition" and "Run Partition" display the configured number of cache partitions and the number of partitions currently in use. These may differ if you have changed the number of partitions since last reboot.
- "APF Percent" displays the percentage of the pool that can hold unused buffers brought in by asynchronous prefetch.

A summary line prints the total size of the cache or caches displayed.

Configuring data caches

The default data cache and the procedure cache for Adaptive Server are specified using an absolute value. The first step in planning cache configuration and implementing caches is to set the max memory configuration parameter. After you set max memory, determine how much space you want to allocate for data caches on your server. The size of a data cache is limited only by access to memory on the system; however, max memory should be larger than total logical memory. You must specify an absolute value for the size of the default data cache and all other user-defined caches. For an overview of Adaptive Server memory usage, see Chapter 18, “Configuring Memory.”

You can configure data caches in two ways:

- Interactively, using sp_cacheconfig and sp_poolconfig
- By editing your configuration file

The following sections describe how to use sp_cacheconfig and sp_poolconfig. See “Configuring data caches with the configuration file” on page 628 for information about using the configuration file.

Each time you execute sp_cacheconfig or sp_poolconfig, Adaptive Server writes the new cache or pool information into the configuration file and copies the old version of the file to a backup file. A message giving the backup file name is sent to the error log.

The syntax to create a new cache is:

```
sp_cacheconfig cache_name, "size[P|K|M|G]"
```

Size units can be specified with:

- P – pages (Adaptive Server logical page size)
- K – kilobytes (default)
- M – megabytes
- G – gigabytes

Maximum data cache size is limited only by the amount of memory available on your system.

This command configures a 10MB cache named pubs_cache:

```
sp_cacheconfig pubs_cache, "10M"
```

This command makes changes in the system tables and writes the new values to the configuration file, but does not activate the cache. You must restart Adaptive Server for the changes to take effect.

Using sp_cacheconfig to see the configuration before a restart shows different “Config” and “Run” values:

```
sp_cacheconfig pubs_cache

Cache Name                Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ------------
pubs_cache                Pend/Act  Mixed       10.00 Mb      0.00 Mb
                                             ------------ ------------
                                      Total     10.00 Mb      0.00 Mb
```

The status “Pend/Act” for pubs_cache shows that the configuration of this cache is pending, waiting for a restart. “Config Value” displays 10MB, and “Run Value” displays 0. Run values and configuration values are also different when you delete caches and when you change their size.

The section of output that provides detail about pools does not print for caches that are not active.

After a restart of Adaptive Server, sp_cacheconfig reports:

```
sp_cacheconfig

Cache Name                Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ------------
default data cache        Active    Default      0.00 Mb     49.37 Mb
pubs_cache                Active    Mixed       10.00 Mb     10.00 Mb
                                             ------------ ------------
                                      Total     10.00 Mb     59.37 Mb
======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 0.00 Mb,   Run Size: 49.37 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb  10110 Kb      0.00 Mb     49.37 Mb     10
======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
```

```
-------- --------- ------------ ----------- -----------
    2 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
```

The pubs_cache is now active, and all the space is assigned to the pool. The size of the default cache has been reduced by 10MB. The remainder of the difference in the size of the default cache and the total amount of cache available is due to changing overhead values.

Here is the same example on a 16K server:

```
1> sp_cacheconfig pubs_cache
Cache Name                     Status    Type     Config Value Run Value
------------------------------ --------- -------- ------------ ------------
pubs_cache                     Active    Mixed       10.00 Mb     10.00 Mb
------------ ------------
                                            Total      10.00 Mb     10.00 Mb
==========================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
       Config Size: 10.00 Mb,   Run Size: 10.00 Mb
       Config Replacement: strict LRU,   Run Replacement: strict LRU
       Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ----------- -----------
16 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
```

See "How overhead affects total cache space" on page 612 for examples.

You can create as many caches as you want before restarting Adaptive Server. You must restart Adaptive Server before you can configure pools or bind objects to newly created caches.

Explicitly configuring the default cache

You must explicitly configure the size of the default data cache because it is specified with an absolute value. Use sp_helpcache to see the amount of memory remaining that can be used for the cache. For example:

```
sp_helpcache
Cache Name                Config Size     Run Size       Overhead
------------------------ -------------   ----------     ----------
default data cache         25.00 Mb       25.00 Mb        0.22 Mb
pubs_cache                 10.00 Mb       10.00 Mb        0.11 Mb
pubs_log                   31.25 Mb       31.25 Mb        0.27 Mb
```

```
Memory Available For      Memory Configured
Named Caches              To Named Caches
--------------------       ----------------
66.44 Mb                    66.25 Mb

------------------ Cache Binding Information: ------------------

Cache Name   Entity Name Type    Index Name              Status
----------   ----------------    ----------              ----------
```

To specify the absolute size of the default data cache, execute sp_cacheconfig with default data cache and a size value. This command sets the default data cache size to 25MB:

```
sp_cacheconfig "default data cache", "25M"
```

After a restart of the server, “Config Value” shows the value.

```
sp_cacheconfig

Cache Name                Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ------------
default data cache        Active    Default      25.00 Mb     49.37 Mb
pubs_cache                Active    Mixed        10.00 Mb     10.00 Mb
                                             ------------ ------------
                                       Total     10.00 Mb     59.37 Mb
======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 49.37 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb  10110 Kb     00.00 Mb     49.37 Mb     10
======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
```

You can change the size of the default data cache by using sp_cacheconfig. Any changes made to the size of the default data cache will not affect the size of any other cache. Similarly, once the size of the default data cache is specified, the configuration of other user-defined caches does not alter the size of the default data cache. When a user-defined cache is created, the memory is taken from max memory, without changing the size of the default data cache.

Note If you configure the default data cache and then reduce max memory to a level that sets the total logical memory value higher than the max memory value, Adaptive Server will not start. Edit your configuration file to increase the size of other caches and increase the values of configuration parameters that require memory to create an environment in which total logical memory is higher than max memory. See Chapter 18, "Configuring Memory," for more information.

The default data cache and all user-defined caches are explicitly configured with an absolute value. In addition, many configuration parameters use memory. To maximize performance and avoid errors, set the value of max memory to a level high enough to accommodate all caches and all configuration parameters that use memory.

Adaptive Server issues a warning message if you set max memory to a value less than total logical memory.

Changing the cache type

To reserve a cache for use by only the transaction log, change the cache's type to "logonly." This example creates the cache pubs_log with the type "logonly:"

```
sp_cacheconfig pubs_log, "7M", "logonly"
```

This shows the state of the cache before a restart:

```
Cache Name          Status    Type     Config Value Run Value
------------------- --------- -------- ------------ ------------
pubs_log            Pend/Act  Log Only      7.00 Mb      0.00 Mb
                                       ------------ ------------
                                Total      7.00 Mb      0.00 Mb
```

You can change the type of an existing "mixed" cache, as long as no non-log objects are bound to it:

```
sp_cacheconfig pubtune_cache, logonly
```

In high transaction environments, Adaptive Server usually performs best if you allocate two logical pages to the transaction logs. For larger page sizes (4, 8, and 16K) use sp_sysmon to find the optimum configuration for your site. For information on configuring caches for improved log performance, see "Matching log I/O Size for log caches" on page 609.

Configuring cache replacement policy

If a cache is dedicated to a table or an index, and the cache has little or no buffer replacement when the system reaches a stable state, you can set relaxed LRU (least recently used) replacement policy. Relaxed LRU replacement policy can improve performance for caches where there is little or no buffer replacement occurring, and for most log caches. See Chapter 32, "Memory Use and Performance," in the *Performance and Tuning Guide* for more information. To set relaxed replacement policy, use:

```
sp_cacheconfig pubs_log, relaxed
```

The default value is "strict."

You can create a cache and specify its cache type and the replacement policy in one command. These examples create two caches, pubs_log and pubs_cache:

```
sp_cacheconfig pubs_log, "3M", logonly, relaxed
sp_cacheconfig pubs_cache, "10M", mixed, strict
```

You must restart Adaptive Server for cache replacement policy changes to take effect.

Here are the results after a restart:

```
sp_cacheconfig

Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     42.29 Mb
pubs_cache               Active    Mixed        10.00 Mb     10.00 Mb
pubs_log                 Active    Log Only      7.00 Mb      7.00 Mb
                                            ------------ ------------
                                     Total      42.00 Mb     59.29 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb      0.00 Mb     42.29 Mb     10
=======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 7.00 Mb,   Run Size: 7.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   1432 Kb      0.00 Mb      7.00 Mb     10
```

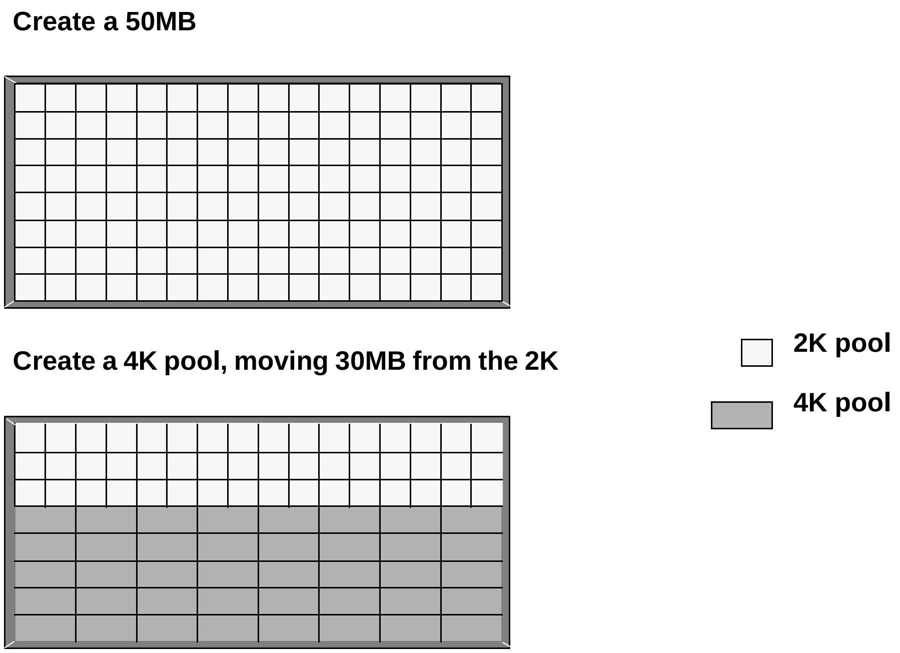
Dividing a data cache into memory pools

After you create a data cache, you can divide it into memory pools, each with a different I/O size. In any cache, you can have only one pool of each I/O size. The minimum size you can create a memory pool is the size of the server’s logical page. Memory pools larger than this must be a power of two and can be a maximum size of one extent.

When Adaptive Server performs large I/Os, multiple pages are read into the cache at the same time. These pages are always treated as a unit; they age in the cache and are written to disk as a unit.

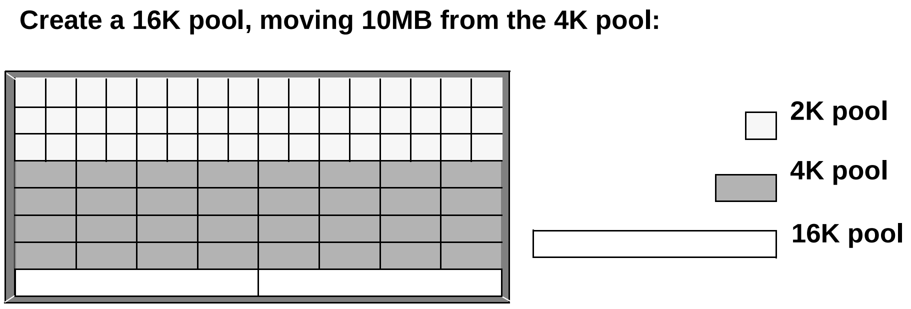
By default, when you create a named data cache, all of its space is assigned to the default memory pool. Creating additional pools reassigns some of that space to other pools, reducing the size of the default memory pool. For example, if you create a data cache with 50MB of space, all the space is assigned to the 2K pool. If you configure a 4K pool with 30MB of space in this cache, the 2K pool is reduced to 20MB.

Figure 19-2: Configuring a cache and a 4K memory pool



After you create the pools, you can move space between them. For example, in a cache with a 20MB 2K pool and a 30MB 4K pool, you can configure a 16K pool, taking 10MB of space from the 4K pool.

Figure 19-3: Moving space from an existing pool to a new pool



The commands that move space between pools within a cache do not require a restart of Adaptive Server, so you can reconfigure pools to meet changing application loads with little impact on server activity.

In addition to creating pools in the caches you configure, you can add memory pools for I/Os up to 16K to the default data cache.

The syntax for configuring memory pools is:

```
sp_poolconfig cache_name, "memsize[P|K|M|G]",
"config_poolK" [, "affected_poolK"]
```

Pool configuration sets the config_pool to the size specified in the command. It always affects a second pool (the affected_pool) by moving space to or from that pool. If you do not specify the affected_pool, the space is taken from or allocated to the 2K pool. The minimum size for a pool is 512K.

This example creates a 7MB pool of 16K pages in the pubs_cache data cache:

```
sp_poolconfig pubs_cache, "7M", "16K"
```

This command reduces the size of the memory pool. To see the current configuration, run sp_cacheconfig, giving only the cache name:

```
sp_cacheconfig pubs_cache

Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_cache               Active    Mixed        10.00 Mb     10.00 Mb
                                            ------------ ------------
                                     Total      10.00 Mb     10.00 Mb
=======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb      0.00 Mb      3.00 Mb     10


   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
```

You can also create memory pools in the default data cache.

In the following example, you start with this cache configuration:

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     42.29 Mb
                                            ------------ ------------
                                     Total      25.00 Mb     42.29 Mb
```

```
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb      0.00 Mb     42.29 Mb     10
```

This command creates a 16K pool in the default data cache that is 8 megabytes:

```
sp_poolconfig "default data cache", "8M", "16K"
```

It results in this configuration, reducing the “Run Size” of the 2K pool:

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     42.29 Mb
                                            ------------ ------------
                                     Total      25.00 Mb     42.29 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb      0.00 Mb     34.29 Mb     10
   16 Kb   1632 Kb      8.00 Mb      8.00 Mb     10
```

You do not need to configure the size of the 2K memory pool in caches that you create. Its “Run Size” represents all the memory not explicitly configured to other pools in the cache.

Matching log I/O Size for log caches

If you create a cache for the transaction log of a database, configure most of the space in that cache to match the log I/O size. The default value is twice the server's logical page size (for a server with 2K logical page size, it is 4K, for a server with 4K logical page size, it is 8K, and so on). Adaptive Server uses 2K I/O for the log if a 4K pool is not available. The log I/O size can be changed with sp_logiosize. The log I/O size of each database is reported in the error log when Adaptive Server starts, or you can check the size of a database by using the database and issuing sp_logiosize with no parameters.

This example creates a 4K pool in the pubs_log cache:

```
sp_poolconfig pubs_log, "3M", "4K"
```

You can also create a 4K memory pool in the default data cache for use by transaction logs of any databases that are not bound to another cache:

```
sp_poolconfig "default data cache", "2.5M", "4K"
```

See Chapter 32, "Choosing the I/O Size for the Transaction Log," in the *Performance and Tuning Guide* for information on tuning the log I/O size.

Binding objects to caches

sp_bindcache assigns a database, table, index or text/image object to a cache. Before you can bind an entity to a cache, the following conditions must be met:

- The named cache must exist, and its status must be "Active."
- The database or database object must exist.
- To bind tables, indexes, or objects, you must be using the database where they are stored.
- To bind system tables, including the transaction log table syslogs, the database must be in single-user mode.
- To bind a database, you must be using master, and the database must be in single-user mode.
- To bind a database, user table, index, text object, or image object to a cache, the type of cache must be "Mixed." Only the syslogs table can be bound to a cache of "Log Only" type.

- You must own the object or be the Database Owner or the System Administrator.

To bind objects to caches, you must restart Adaptive Server after creating caches. However, bindings take effect immediately and do not require a restart.

The syntax for binding objects to caches is:

```
sp_bindcache cache_name, dbname [,[owner.]tablename
[, indexname |  "text only" ] ]
```

The owner name is optional if the table is owned by "dbo."

This command binds the titles table to the pubs_cache:

```
sp_bindcache  pubs_cache, pubs2, titles
```

To bind an index on titles, add the index name as the third parameter:

```
sp_bindcache pubs_cache, pubs2,  titles, titleind
```

The owner name is not needed in the examples above because the objects in the pubs2 database are owned by "dbo." To specify a table owned by any other user, add the owner name. You must enclose the parameter in quotation marks, since the period in the parameter is a special character:

```
sp_bindcache pubs_cache, pubs2, "fred.sales_east"
```

This command binds the transaction log, syslogs, to the pubs_log cache:

```
sp_bindcache pubs_log, pubs2, syslogs
```

The database must be in single-user mode before you can bind any system tables, including the transaction log, syslogs, to a cache. Use sp_dboption from master, and a use database command, and run checkpoint:

```
sp_dboption pubs2, single, true
use pubs2
checkpoint
```

text and image columns for a table are stored in a separate data structure in the database. To bind this object to a cache, add the "text-only" parameter:

```
sp_bindcache pubs_cache, pubs2, au_pix, "text only"
```

This command, executed from master, binds the tempdb database to a cache:

```
sp_bindcache tempdb_cache, tempdb
```

You can rebind objects without dropping existing bindings.

Cache binding restrictions

You cannot bind or unbind a database object when:

- Dirty reads are active on the object.
- A cursor is open on the objec.t

In addition, Adaptive Server needs to lock the object while the binding or unbinding takes place, so the procedure may have a slow response time, because it waits for locks to be released. See“Locking to perform bindings” on page 627 for more information.

Getting information about cache bindings

sp_helpcache provides information about a cache and the entities bound to it when you provide the cache name:

```
sp_helpcache pubs_cache

Cache Name           Config Size     Run Size       Overhead
-------------------- -------------   ----------     ----------
pubs_cache             10.50 Mb      10.50 Mb        0.56 Mb

------------------ Cache Binding Information: ------------------

Cache Name      Entity Name           Type   Index Name  Status
----------      -----------           ----   ----------  ------
pubs_cache      pubs2.dbo.titles      index  titleind    V
pubs_cache      pubs2.dbo.au_pix      index  tau_pix     V
pubs_cache      pubs2.dbo.titles      table              V
pubs_cache      pubs2.fred.sales_east table              V
```

If you use sp_helpcache without a cache name, it prints information about all the configured caches on Adaptive Server and all the objects that are bound to them.

sp_helpcache performs string matching on the cache name, using *%cachename%*. For example, “pubs” matches both “pubs_cache” and “pubs_log.”

The “Status” column reports whether a cache binding is valid (“V”) or invalid (“I”). If a database or object is bound to a cache, and the cache is deleted, binding information is retained in the system tables, but the cache binding is marked as invalid. All objects with invalid bindings use the default data cache. If you subsequently create another cache with the same name, the binding becomes valid when the cache is activated by a restart of Adaptive Server.

Checking cache overhead

sp_helpcache can report the amount of overhead required to manage a named data cache of a given size. When you create a named data cache, all the space you request with sp_cacheconfig is made available for cache space. The memory needed for cache management is taken from the default data cache.

To see the overhead required for a cache, give the proposed size. You can use P for pages, K for kilobytes, M for megabytes, or G for gigabytes. The following example checks the overhead for 20,000 pages:

```
sp_helpcache "20000P"
```

```
2.08Mb of overhead memory will be needed to manage a
cache of size 20000P
```

You are not wasting any cache space by configuring user caches. About 5% of memory is required for the structures that store and track pages in memory, whether you use a single large data cache or several smaller caches.

How overhead affects total cache space

The example detailed in “Information on data caches” on page 596 shows a default data cache with 59.44 MB of cache space available before any user-defined caches are created. The server in this example uses a 2K logical page. When the 10MB pubs_cache is created and Adaptive Server is restarted, the results of sp_cacheconfig show a total cache size of 59.44 MB.

Configuring a data cache can appear to increase or decrease the total available cache. The explanation for this lies in the amount of overhead required to manage a cache of a particular size, and the fact that the overhead is not included in the values displayed by sp_cacheconfig.

Using sp_helpcache to check the overhead of the original 59.44MB default cache and the new 10MB cache shows that the change in space is due to changes in the size of overhead. The following command shows the overhead for the default data cache before any changes were made:

```
sp_helpcache "59.44M"
```

```
3.04Mb of overhead memory will be needed to manage a
cache of size 59.44M
```

This command shows the overhead for pubs_cache:

```
sp_helpcache "10M"
```

```
0.53Mb of overhead memory will be needed to manage a
cache of size 10M
```

The following calculations add the overhead required to manage the original cache space and then subtract the overhead for pubs_cache:

Original total cache size (overhead not included)	**59.44**
Overhead for 59.44 MB default cache	+3.04
Total cache space, including overhead	62.48
10MB pubs_cache and .53MB overhead	-10.53
Remaining space	51.95
Overhead for 51.95MB cache	- 2.69
Usable size of the default cache	49.26

Cache sizes are rounded to two places when printed by sp_cacheconfig, and overhead is rounded to two places by sp_helpcache, so you will see a small amount of rounding error in the output.

Dropping cache bindings

Two commands drop cache bindings:

- sp_unbindcache unbinds a single entity from a cache.
- sp_unbindcache_all unbinds all objects bound to a cache.

The syntax for sp_unbindcache is:

```
sp_unbindcache dbname [,[owner.]tablename
```

```
    [, indexname | "text only"] ]
```

This commands unbinds the titleidind index on the titles table in the pubs2 database:

```
    sp_unbindcache pubs2, titles, titleidind
```

To unbind all the objects bound to a cache, use sp_unbindcache_all, giving the cache's name:

```
    sp_unbindcache_all pubs_cache
```

Note You cannot use sp_unbindcache_all if more than eight databases and/or objects in eight databases are bound to the cache. You must use sp_unbindcache on individual databases or objects to reduce the number of databases involved to eight or less.

When you drop a cache binding for an object, all the pages currently in memory are cleared from the cache.

Changing the wash area for a memory pool

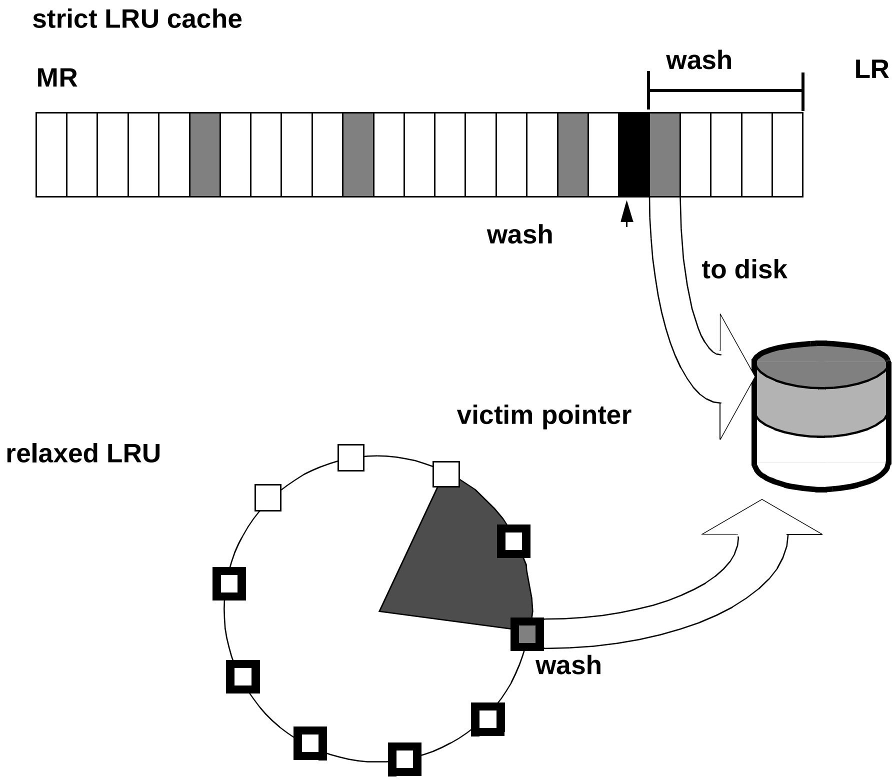
When Adaptive Server needs to read a buffer into cache, it places the buffer:

- At the LRU (least recently used) end of each memory pool, in a cache with strict LRU policy
- At the victim pointer, in a cache with relaxed LRU policy. If the recently used bit of buffer at the victim marker is set, the victim pointer is moved to the next buffer in the pool.

A portion of each pool is configured as the **wash area**. After dirty pages (pages that have been changed in cache) pass the wash marker and enter the wash area, Adaptive Server starts an asynchronous I/O on the page. When the write completes, the page is marked clean and remains available in the cache.

The space in the wash area must be large enough so that the I/O on the buffer can complete before the page needs to be replaced. Figure 19-4 illustrates how the wash area of a buffer pool works with a strict and relaxed LRU cache.

Figure 19-4: Wash area of a buffer pool



By default, the size of the wash area for a memory pool is configured as follows:

- If the pool size is less than 300MB, the default wash size is 20 percent of the buffers in the pool.
- If the pool size is greater than 300MB, the default wash size is 20 percent of the number of buffers in 300MB.

The minimum wash size is 10 buffers. The maximum size of the wash area is 80 percentof the pool size.

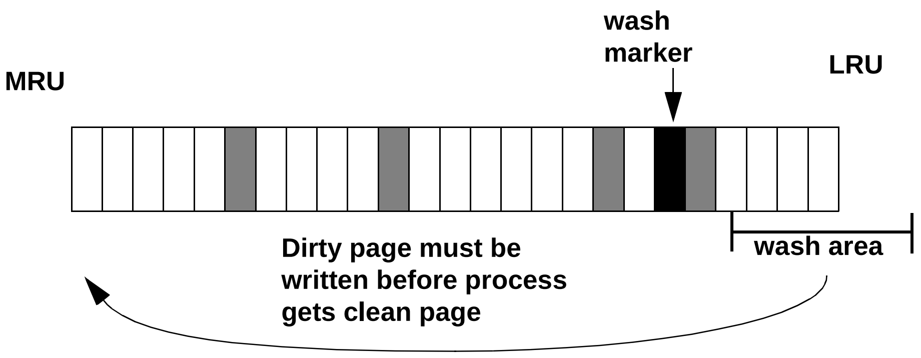
A buffer is a block of pages that matches the I/O size for the pool. Each buffer is treated as a unit: all pages in the buffer are read into cache, written to disk, and aged in the cache as a unit. For the size of the block, multiply the number of buffers by the pool size—for a 2K pool, 256 buffers equals 512K; for a 16K pool, 256 buffers equals 4096K.

For example, if you configure a 16K pool with 1MB of space, the pool has 64 buffers; 20 percent of 64 is 12.8. This is rounded down to 12 buffers, or 192K, are allocated to the wash area.

When the wash area is too small

If the wash area is too small for the usage in a buffer pool, operations that need a clean buffer may have to wait for I/O to complete on the dirty buffer at the LRU end of the pool or at the victim marker. This is called a **dirty buffer grab**, and it can seriously impact performance. Figure 19-5 shows a dirty buffer grab on a strict replacement policy cache.

Figure 19-5: Small wash area results in a dirty buffer grab



You can use sp_sysmon to determine whether dirty buffer grabs are taking place in your memory pools. Run sp_sysmon while the cache is experiencing a heavy period of I/O and heavy update activity, since it is the combination of many dirty pages and high cache replacement rates that usually causes dirty buffer grabs.

If the “Buffers Grabbed Dirty” output in the cache summary section shows a nonzero value in the “Count” column, check the “Grabbed Dirty” row for each pool to determine where the problem lies. Increase the size of the wash area for the affected pool. This command sets the wash area of the 8K memory pool to 720K:

```
sp_poolconfig pubs_cache, "8K", "wash=720K"
```

If the pool is very small, you may also want to increase its pool size, especially if sp_sysmon output shows that the pool is experiencing high turnover rates.

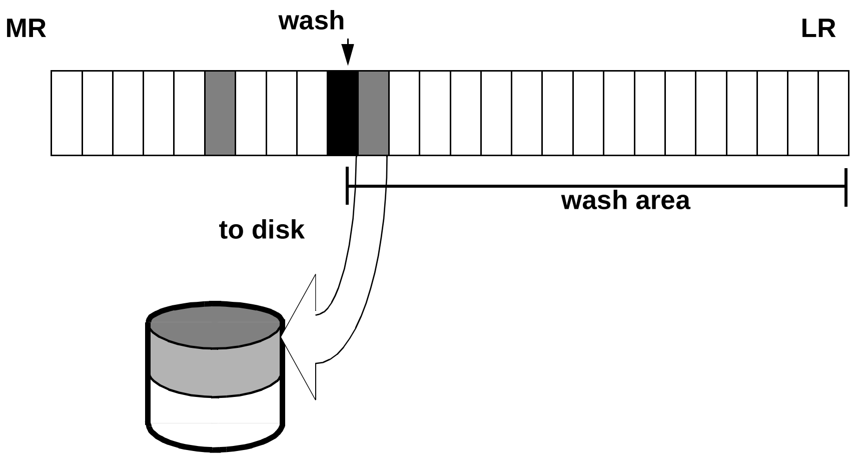
See the *Performance and Tuning Guide* for more information.

When the wash area is too large

If the wash area is too large in a pool, the buffers move too quickly past the "wash marker" in cache, and an asynchronous write is started on any dirty buffers, as shown in Figure 19-6 . The buffer is marked "clean" and remains in the wash area of the MRU/LRU chain until it reaches the LRU. If another query changes a page in the buffer, Adaptive Server must perform additional I/O to write it to disk again.

If sp_sysmon output shows a high percentage of buffers "Found in Wash" for a strict replacement policy cache, and there are no problems with dirty buffer grabs, you may want to try reducing the size of the wash area. See the *Performance and Tuning Guide* for more information.

Figure 19-6: Effects of making the wash area too large



Changing the asynchronous prefetch limit for a pool

The asynchronous prefetch limit specifies the percentage of the pool that can be used to hold pages that have been brought into the cache by asynchronous prefetch, but have not yet been used by any queries. The default value for the server is set with the global async prefetch limit configuration parameter. Pool limits, set with sp_poolconfig, override the default limit for a single pool.

This command sets the percentage for the 2K pool in the pubs_cache to 20:

```
sp_poolconfig pubs_cache, "2K", "local async prefetch limit=20"
```

Changes to the prefetch limit for a pool take effect immediately and do not require a restart of Adaptive Server. Valid values are 0–100. Setting the prefetch limit to 0 disables asynchronous prefetching in a pool. For information about the impact of asynchronous prefetch on performance, see Chapter 34, “Tuning Asynchronous Prefetch,” in the *Performance and Tuning Guide*.

Resizing named data caches

To change the size of an existing cache, issue sp_cacheconfig, specifying a new total size for the cache. When you increase the size of a cache, all the additional space is added to the pool. When you decrease the size of a cache, all the space is taken from the default pool. You cannot decrease the size of the default pool to less than 512K.

Increasing the size of a cache

sp_cacheconfig reports that pubs_cache is currently configured with 10MB of space:

```
sp_cacheconfig pubs_cache

Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_cache               Active    Mixed        10.00 Mb     10.00 Mb
                                            ------------ ------------
                                     Total      10.00 Mb     10.00 Mb
=======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
```

```
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    720 Kb      0.00 Mb      3.00 Mb     20
   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
```

To increase the size of the cache and its pool, specify the new total size of the cache:

```
sp_cacheconfig pubs_cache, "20M"
```

This output reports the configuration of the pubs_cache before a restart:

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_cache                Active    Mixed       20.00 Mb     10.00 Mb
                                            ------------ ------------
                                      Total     20.00 Mb     10.00 Mb
=======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    720 Kb      0.00 Mb      3.00 Mb     20
   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
```

The additional 10MB has been configured and becomes available in the pool at the next restart.

Decreasing the size of a cache

You can also reduce the size of a cache. For example, following is a report on the pubs_log cache:

```
sp_cacheconfig pubs_log
```

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_log                  Active    Log Only      7.00 Mb      7.00 Mb
                                            ------------ ------------
                                      Total      7.00 Mb      7.00 Mb
```

```
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 7.00 Mb,   Run Size: 7.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:           1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    920 Kb      0.00 Mb      4.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

The following command reduces the size of the pubs_log cache, reducing the size of the default pool:

```
sp_cacheconfig pubs_log, "6M"
```

After a restart of Adaptive Server, sp_cacheconfig shows:

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
                                            ------------ ------------
                                     Total      6.00 Mb      6.00 Mb
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:           1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      3.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

When you reduce the size of a data cache, all the space to be removed must be available in the pool. You may need to move space to the default pool from other pools before you can reduce the size of the data cache. In the last example, if you wanted to reduce the size of the cache to 3MB, you would need to use sp_poolconfig to move some memory into the default pool of 2K from the 4K pool. The memory is moved to “memory available for named caches”. See “Changing the size of memory pools” on page 621 for more information.

Dropping data caches

To completely remove a data cache, reset its size to 0:

```
sp_cacheconfig pubs_log, "0"
```

This changes the cache status to “Pend/Del.” You must restart Adaptive Server for the change to take effect. Until you do, the cache remains active, and all objects bound to the cache still use it for I/O.

If you delete a data cache, and there are objects bound to the cache, the cache bindings are marked invalid at the next restart of Adaptive Server. All objects with invalid cache bindings use the default data cache. Warning messages are printed in the error log when the bindings are marked invalid. For example, if the titles table in the pubs2 database is bound to a cache, and that cache is dropped, the message in the log is:

```
Cache binding for database ’5’, object ’208003772’,
index ’0’ is being marked invalid in Sysattributes.
```

If you re-create the cache and restart Adaptive Server, the bindings are marked valid again.

You cannot drop the default data cache.

Changing the size of memory pools

To change the size of a memory pool, use sp_poolconfig to specify the cache, the new size for the pool, the I/O size of the pool you want to change, and the I/O size of the pool from which the buffers should be taken. If you do not specify the final parameter, all the space is taken from or assigned to the pool.

Moving space from the memory pool

This command checks the current configuration of the pubs_log cache (The output in this example is based on the examples in the previous sections):

```
sp_cacheconfig pubs_log

Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
```

```
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
                                             ------------ ------------
                                     Total       6.00 Mb      6.00 Mb
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      3.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

This command increases the size of the 4K pool to 5MB, moving the required space from the 2K pool:

```
sp_poolconfig pubs_log, "5M", "4K"
```

```
sp_cacheconfig pubs_log
```

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
                                             ------------ ------------
                                     Total       6.00 Mb      6.00 Mb
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      1.00 Mb     10
    4 Kb   1024 Kb      5.00 Mb      5.00 Mb     10
```

Moving space from other memory pools

To transfer space from another pool specify the cache name, a “to” I/O size, and a “from” I/O size. This output shows the current configuration of the default data cache:

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     29.28 Mb
```

```
                                              ------------ ------------
                                     Total      25.00 Mb     29.28 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      0.00 Mb     18.78 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
   16 Kb   1632 Kb      8.00 Mb      8.00 Mb     10
```

The following command increases the size of the 4K pool from 2.5MB to 4MB, taking the space from the 16K pool:

```
sp_poolconfig "default data cache","4M", "4K","16K"
```

This results in the following configuration:

```
Cache Name                Status    Type     Config Value Run Value

------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     29.28 Mb
                                              ------------ ------------
                                     Total      25.00 Mb     29.28 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      0.00 Mb     18.78 Mb     10
    4 Kb    512 Kb      4.00 Mb      4.00 Mb     10
   16 Kb   1632 Kb      6.50 Mb      6.50 Mb     10
```

When you issue a command to move buffers between pools in a cache, Adaptive Server can move only “free” buffers. It cannot move buffers that are in use or buffers that contain changes that have not been written to disk.

When Adaptive Server cannot move as many buffers as you request, it displays an informational message, giving the requested size and the resulting size of the memory pool.

Adding cache partitions

On multi-engine servers, more than one task can attempt to access the cache at the same time. By default, each cache has a single spinlock, so that only one task can change or access the cache at a time. If cache spinlock contention is above 10 percent, increasing the number of cache partitions for a cache can reduce spinlock contention, and improve performance.

You can configure the number of cache partitions for:

- All data caches, using the global cache partition numberconfiguration parameter
- An individual cache, using sp_cacheconfig

The number of partitions in a cache is always a power of 2 between 1 and 64. No pool in any cache partition can be smaller than 512K. In most cases, since caches can be sized to meet requirements for storing individual objects, you should use the local setting for the particular cache where spinlock contention is an issue.

See “Reducing Spinlock Contention with Cache Partitions” on page 32-18 of the *Performance and Tuning Guide* for information on choosing the number of partitions for a cache.

Setting the number of cache partitions with *sp_configure*

Use sp_configure to set the number of cache partitions for all caches on a server. For example, to set the number of cache partitions to 2, enter:

```
sp_configure "global cache partition number",2
```

You must restart the server for the change to take effect.

Setting the number of local cache partitions

Use sp_cacheconfig or the configuration file to set the number of local cache partitions. This command sets the number of cache partitions in the default data cache to 4:

```
sp_cacheconfig "default data cache", "cache_partition=4"
```

You must restart the server for the change to take effect.

Precedence

The local cache partition setting always takes precedence over the global cache partition value.

These commands set the server-wide partition number to 4, and the number of partitions for pubs_cache to 2:

```
sp_configure "global cache partition number", 4
sp_cacheconfig "pubs_cache", "cache_partition=2"
```

The local cache partition number takes precedence over the global cache partition number, so pubs_cache uses 2 partitions. All other configured caches have 4 partitions.

To remove the local setting for pubs_cache, and use the global value instead, use this command:

```
sp_cacheconfig "pubs_cache", "cache_partition=default"
```

To reset the global cache partition number to the default, use:

```
sp_configure "global cache partition number", 0, "default"
```

Dropping a memory pool

To completely remove a pool, reset its size to 0. The following removes the 16K pool and places all space in the default pool:

```
sp_poolconfig "default data cache", "0",  "16K"
sp_cacheconfig "default data cache"
```

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache       Active    Default      25.00 Mb     29.28 Mb
                                            ------------ ------------
                                     Total      25.00 Mb     29.28 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:           1,   Run Partition:           1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      6.50 Mb     25.28 Mb     10
    4 Kb    512 Kb      4.00 Mb      4.00 Mb     10
```

If you do not specify the affected pool size (16K in the example above), all the space is placed in the default pool. You cannot delete the default pool in any cache.

When pools cannot be dropped due to pages use

If the pool you are trying to delete contains pages that are in use, or pages that have dirty reads, but are not written to disk, Adaptive Server moves as many pages as possible to the specified pool and prints an informational message telling you the size of the remaining pool. If the pool size is smaller than the minimum allowable pool size, you also receive a warning message saying the pool has been marked unavailable. If you run sp_cacheconfig after receiving one of these warnings, the pool detail section for these pools contains an extra "Status" column, with either "Unavailable/too small" or "Unavailable/deleted" for the affected pool.

You can reissue the command at a later time to complete removing the pool. Pools with "Unavailable/too small" or "Unavailable/deleted" are also removed when you restart Adaptive Server.

Cache binding effects on memory and query plans

Binding and unbinding objects may have an impact on performance. When you bind or unbind a table or an index:

- The object's pages are flushed from the cache.
- The object must be locked to perform the binding.
- All query plans for procedures and triggers must be recompiled.

Flushing pages from cache

When you bind an object or database to a cache, the object's pages that are already in memory are removed from the source cache. The next time the pages are needed by a query, they are read into the new cache. Similarly, when you unbind objects, the pages in cache are removed from the user-configured cache and read into the default cache the next time they are needed by a query.

Locking to perform bindings

To bind or unbind user tables, indexes, or text or image objects, the cache binding commands need an exclusive table lock on the object. If a user holds locks on a table, and you issue sp_bindcache, sp_unbindcache, or sp_unbindcache_all on the object, the system procedure sleeps until it can acquire the locks it needs.

For databases, system tables, and indexes on system tables, the database must be in single-user mode, so there cannot be another user who holds a lock on the object.

Cache binding effects on stored procedures and triggers

Cache bindings and I/O sizes are part of the query plan for stored procedures and triggers. When you change the cache binding for an object, all the stored procedures that reference the object are recompiled the next time they are executed. When you change the cache binding for a database, all stored procedures that reference any objects in the database that are not explicitly bound to a cache are recompiled the next time they are run.

Configuring data caches with the configuration file

You can add or drop named data caches and reconfigure existing caches and their memory pools by editing the configuration file that is used when you start Adaptive Server.

Note You cannot reconfigure caches and pools on a server while it is running. Any attempt to read a configuration file that contains cache and pool configurations different from those already configured on the server causes the read to fail.

Cache and pool entries in the configuration file

Each configured data cache on the server has this block of information in the configuration file:

```
[Named Cache:cache_name]
     cache size = {size | DEFAULT}
     cache status = {mixed cache | log only | default data cache}
     cache replacement policy = {DEFAULT |
         relaxed LRU replacement| strict LRU replacement }
```

Size units can be specified with:

- P – pages (Adaptive Server pages)
- K – kilobytes (default)
- M – megabytes
- G – gigabytes

This example shows the configuration file entry for the default data cache:

```
[Named Cache:default data cache]
      cache size = DEFAULT
      cache status = default data cache
      cache replacement policy = strict LRU replacement
```

The default data cache entry is the only cache entry that is required in for Adaptive Server to start. It must have the cache size and cache status, and the status must be “default data cache.”

If the cache has pools configured in addition to the pool, the block in the preceding example is followed by a block of information for each pool:

```
[16K I/O Buffer Pool]
        pool size = size
        wash size = size
        local async prefetch limit = DEFAULT
```

Note In some cases, there is no configuration file entry for the pool in a cache. If you change the asynchronous prefetch percentage with sp_poolconfig, the change is not written to the configuration file, only to system tables.

This example shows output from sp_cacheconfig, followed by the configuration file entries that match this cache and pool configuration:

```
 Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
default data cache        Active    Default      25.00 Mb     29.28 Mb
pubs_cache                Active    Mixed        20.00 Mb     20.00 Mb
pubs_log                  Active    Log Only      6.00 Mb      6.00 Mb
tempdb_cache              Active    Mixed         4.00 Mb      4.00 Mb
                                             ------------ ------------
                                     Total       55.00 Mb     59.28 Mb
=======================================================================
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      6.50 Mb     25.28 Mb     10
    4 Kb    512 Kb      4.00 Mb      4.00 Mb     10
=======================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 20.00 Mb,   Run Size: 20.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2662 Kb      0.00 Mb     13.00 Mb     10
   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
=======================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      1.00 Mb     10
    4 Kb   1024 Kb      5.00 Mb      5.00 Mb     10
=======================================================================
Cache: tempdb_cache,   Status: Active,   Type: Mixed
     Config Size: 4.00 Mb,   Run Size: 4.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    818 Kb      0.00 Mb      4.00 Mb     10
```

This is the matching configuration file information:

```
[Named Cache:default data cache]
        cache size = 25M
        cache status = default data cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT

[2K I/O Buffer Pool]
        pool size = 6656.0000k
        wash size = 3844 K
        local async prefetch limit = DEFAULT

[4K I/O Buffer Pool]
        pool size = 4.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Named Cache:pubs_cache]
        cache size = 20M
        cache status = mixed cache
        cache replacement policy = strict LRU
replacement
        local cache partition number = DEFAULT

[16K I/O Buffer Pool]
        pool size = 7.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Named Cache:pubs_log]
```

```
        cache size = 6M
        cache status = log only
        cache replacement policy = relaxed LRU
replacement
        local cache partition number = DEFAULT

[4K I/O Buffer Pool]
        pool size = 5.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT


[Named Cache:tempdb_cache]
        cache size = 4M
        cache status = mixed cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT
```

For more information about the configuration file, see Chapter 7, "Setting Configuration Parameters."

Warning! Check the max memory configuration parameter and allow enough memory for other Adaptive Server needs. If you assign too much memory to data caches in your configuration file, Adaptive Server will not start. If this occurs, edit the configuration file to reduce the amount of space in the data caches, or increase the max memory allocated to Adaptive Server. See Chapter 18, "Configuring Memory," for suggestions on monitoring cache sizes.

Cache configuration guidelines

User-definable caches are a performance feature of Adaptive Server. This chapter addresses only the mechanics of configuring caches and pools and binding objects to caches. Performance information and suggested strategies for testing cache utilization is addressed in Chapter 32, "Memory Use and Performance," in the *Performance and Tuning Guide*.

Here are some general guidelines:

- The size of the default data cache does not decrease when you configure other caches.

- Make sure that your default data cache is large enough for all cache activity on unbound tables and indexes. All objects that are not explicitly bound to a cache use the default cache. This includes any unbound system tables in the user databases, the system tables in master, and any other objects that are not explicitly bound to a cache.
- During recovery, only the 2K memory pool of the default cache is active. Transactions logs are read into the 2K pool of the default cache. All transactions that must be rolled back or rolled forward must read data pages into the default data cache. If the default data cache is too small, it can slow recovery time.
- Do not "starve" the 2K pool in any cache. For many types of data access, there is no need for large I/O. For example, a simple query that uses an index to return a single row to the user might use 4 or 5 2K I/Os, and gain nothing from 16K I/O.
- Certain commands can perform only 2K I/O: disk init, certain dbcc commands, and drop table. dbcc checktable can perform large I/O, and dbcc checkdb performs large I/O on tables and 2K I/O on indexes.
- For caches used by transaction logs, configure an I/O pool that matches the default log I/O size. This size is set for a database using sp_logiosize. The default value is 4K.
- Trying to manage every index and object and its caching can waste cache space. If you have created caches or pools that are not optimally used by the tables or indexes bound to them, they are wasting space and creating additional I/O in other caches.
- If tempdb is used heavily by your applications, bind it to its own cache. Note that you can bind only the entire tempdb database, you cannot bind individual objects from tempdb.
- For caches with high update and replacement rates, be sure that your wash size is large enough.
- On multi-CPU systems, spread your busiest tables and their indexes across multiple caches to avoid spinlock contention.
- Consider reconfiguring caches or the memory pools within caches to match changing workloads. Reconfiguring caches requires a restart of the server, but memory pool reconfiguration does not.

For example, if your system performs mostly OLTP (online transaction processing) during most of the month, and has heavy DSS (decision-support system) activity for a few days, consider moving space from the 2K pool to the 16K pool for the high DSS activity and resizing the pools for OLTP when the DSS workload ends.

Configuration file errors

If you edit your configuration file manually, check the cache, pool, and wash sizes carefully. Certain configuration file errors can cause start-up failure:

- The total size of all of the caches cannot be greater than the amount of max memory, minus other Adaptive Server memory needs.
- The total size of the pools in any cache cannot be greater than the size of the cache.
- The wash size cannot be too small (less than 20 percent of the pool size, with a minimum of 10 buffers) and cannot be larger than 80 percent of the buffers in the pool.
- The default data cache status must be "default data cache," and the size must be specified, either as a numeric value or as "DEFAULT".
- The status and size for any cache must be specified.
- The pool size and wash size for all pools larger than 2K must be specified.
- The status of all user-defined caches must be "mixed cache" or "log only".
- The cache replacement policy and the asynchronous prefetch percentage are optional, but, if specified, they must have correct parameters or "DEFAULT".

In most cases, problems with missing entries are reported as "unknown format" errors on lines immediately following the entry where the size, status, or other information was omitted. Other errors provide the name of the cache where the error occurred and the type of error. For example, you see this error if the wash size for a pool is specified incorrectly:

```
The wash size for the 4k buffer pool in cache
pubs_cache has been incorrectly configured. It must
be a minimum of 10 buffers and a maximum of 80
percent of the number of buffers in the pool.
```

CHAPTER 20

Managing Multiprocessor Servers

This chapter provides guidelines for administering Adaptive Server on a multiprocessor.

Topics covered in this chapter include:

Parallel processing

Adaptive Server implements the Sybase Virtual Server Architecture™, which enables it to take advantage of the parallel processing feature of symmetric multiprocessing (SMP) systems. You can run Adaptive Server as a single process or as multiple, cooperating processes, depending on the number of CPUs available and the demands placed on the server machine. This chapter describes:

- The target machine architecture for the SMP Adaptive Server
- Adaptive Server architecture for SMP environments
- Adaptive Server task management in the SMP environment
- Managing multiple engines

For information on application design for SMP systems, see Chapter 3, “Using Engines and CPUs,” in the *Performance and Tuning Guide*.

Definitions

Here are the definitions for several terms used in this chapter:

- **Process** – an execution environment scheduled onto physical CPUs by the operating system.
- **Engine** – a process running an Adaptive Server that communicates with the other Adaptive Server processes via shared memory. An engine can be thought of as one CPU’s worth of processing power. It does *not* represent a particular CPU. Also referred to as a **server engine**.
- **Task** – an execution environment within the Adaptive Server that is scheduled onto engines by the Adaptive Server.
- **Affinity** – describes a process in which a certain Adaptive Server task runs only on a certain engine (*task affinity*), a certain engine handles network I/O for a certain task (*network I/O affinity*), or a certain engine runs only on a certain CPU (*engine affinity*).
- **Network affinity migration** – describes the process of moving network I/O from one engine to another. SMP systems that support this migration allow Adaptive Server to distribute the network I/O load among all of its engines.

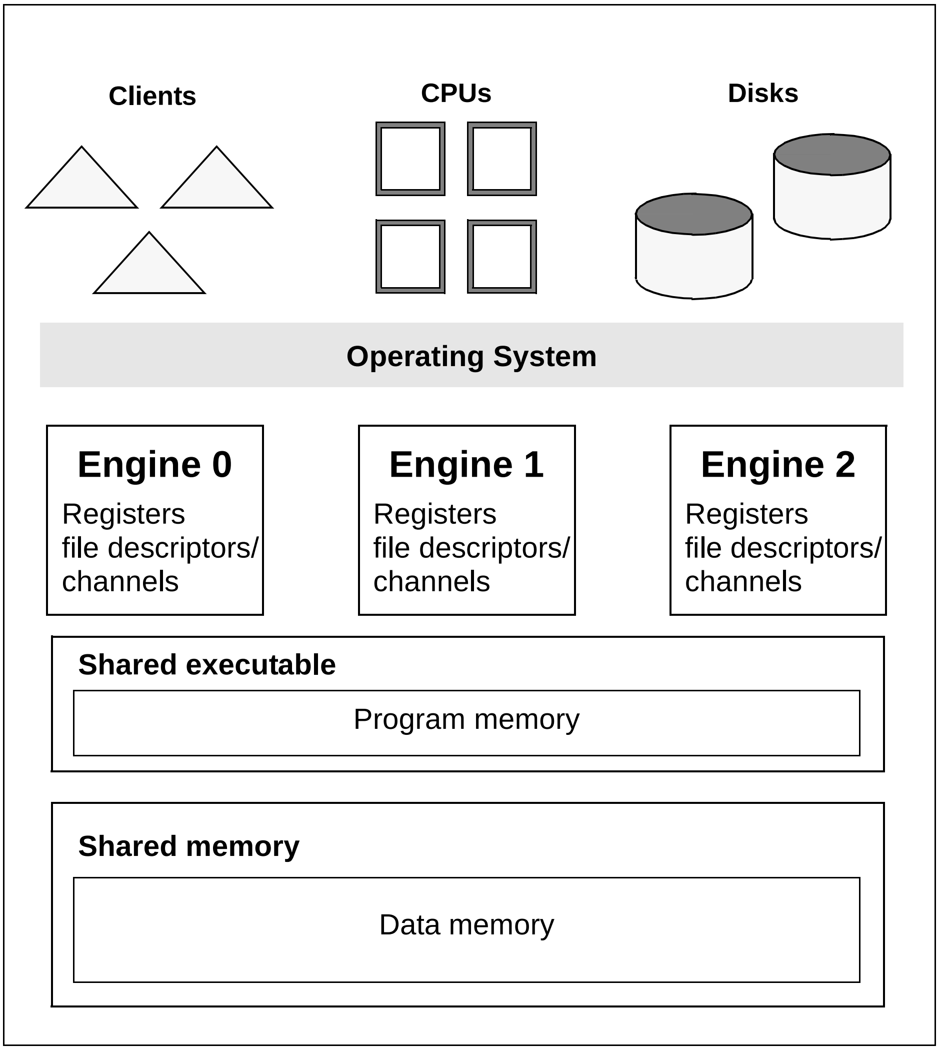
Target architecture

The SMP environment product is intended for machines with the following features:

- A symmetric multiprocessing operating system
- Shared memory over a common bus
- 1–128 processors
- No master processor
- Very high throughput

Adaptive Server consists of one or more cooperating processes (called engines), all of which run the server program in parallel. See Figure 20-1.

Figure 20-1: SMP environment architecture



When clients connect to Adaptive Server, the client connections are assigned to engines in a round-robin fashion, so all engines share the work of handling network I/O for clients. All engines are peers, and they communicate via shared memory.

The server engines perform all database functions, including updates and logging. Adaptive Server, not the operating system, dynamically schedules client tasks onto available engines.

The operating system schedules the engine processes onto physical processors. Any available CPU is used for any engine; there is no *engine affinity*. The processing is called *symmetric* because the lack of affinity between processes and CPUs creates a symmetrically balanced load.

Configuring an SMP environment

Configuring the SMP environment is much the same as configuring the uniprocessor environment, although SMP machines are typically more powerful and handle many more users. The SMP environment provides the additional ability to control the number of engines.

Managing engines

To achieve optimum performance from an SMP system, you must maintain the right number of engines.

An engine represents a certain amount of CPU power. It is a configurable resource like memory.

Note If your server connections use CIS, they are affinitied to a single engine, and will not be allowed to migrate from one engine to another. Adaptive Server uses a load balancing algorithm to evenly distribute the load among the engines.

Resetting the number of engines

When you first install an Adaptive Server, the system is configured for a single engine. To use multiple engines, you must reset the number of engines the first time you restart the server. You may also want to reset the number of engines at other times.

For example:

- You might want to *increase* the number of engines if current performance is not adequate for an application *and* there are enough CPUs on the machine.

- You might want to *decrease* the number of engines if a hardware failure disables CPUs on the machine.

You must restart the server to reset the number of engines.

The max online engines configuration parameter controls the number of engines used by Adaptive Server. Reset this parameter sp_configure. For example, to set the number of engines to 3:

1. Issue the following command:

   ```
   sp_configure "max online engines", 3
   ```

2. Stop and restart the server.

Repeat these steps whenever you need to change the number of engines. Engines other than engine 0 are brought online after recovery is complete.

Choosing the right number of engines

It is important that you choose the right number of engines for Adaptive Server. Here are some guidelines:

- *Never* have more engines than CPUs. Doing so may slow performance. If a CPU goes offline, use sp_configure to reduce the max online engines configuration parameter by 1 and restart Adaptive Server.
- Have only as many engines as you have *usable* CPUs. If there is a lot of processing by the client or other non-Adaptive Server processes, then one engine per CPU may be excessive. Remember, too, that the operating system may take up part of one of the CPUs.
- Have *enough* engines. It is good practice to start with a few engines and add engines when the existing CPUs are almost fully used. If there are too few engines, the capacity of the existing engines will be exceeded and bottlenecks may result.

Taking engines offline with *dbcc engine*

You can dynamically change the number of engines in use by Adaptive Server with the dbcc engine command to take an engine offline or bring an engine online. This allows a System Administrator to reconfigure CPU resources as processing requirements fluctuate over time.

Two configuration parameters limit the number of engines available to the server:

- max online engines – when the server is booted, the number of engines specified by max online engines are started. The number of engines can never exceed max online engines.
- min online engines – sets the minimum number of engines. When you take engines offline using dbcc engine, you cannot reduce the number of engines below the value set by min online engines.

Due to the operating system limit on the number of file descriptors per process, reducing the number of engines reduces the number of network connections that the server can have.

There is no way to migrate a network connection created for server-to-server remote procedure calls, for example, connections to Replication Server and XP Server, so you cannot take an engine offline that is managing one of these connections.

dbcc engine syntax and usage

The syntax for dbcc engine is:

```
dbcc engine(offline , [enginenum] )
dbcc engine("online")
```

If *enginenum* is not specified, the highest-numbered engine is taken offline.

Depending on your operating system and the load on Adaptive Server, taking an engine offline can take several minutes. To complete the task of taking an engine offline, the following steps must be completed:

- All outstanding I/Os for the engine must complete.
- All tasks affiliated with the engine must be migrated to other engines.
- Operating system and internal cleanup must de-allocate all structures.

If tasks cannot be migrated within approximately 5 minutes, the tasks are killed.

Warning! If you use dbcc engine(offline) when CPU utilization is high on the server, Adaptive Server may not be able to migrate all tasks before the time limit expires.Tasks that cannot be migrated within the time limit are killed.

Status and messages during *dbcc engine(offline)*

When a System Administrator issues a dbcc engine(offline) command, messages are sent to the error log. For example, these are the messages on Sun Solaris:

```
00:00000:00000:1999/04/08 15:09:01.13 kernel  engine
5, os pid 19441  offline
```

dbcc engine(offline) returns immediately; you must monitor the error log or check the engine status in sysengines to know that the offline-engine task completes.

An engine with open network connections using Client Library cannot be taken offline. Attempting to offline the engine reports this message in the error log:

```
00:00000:00000:1999/04/08 15:30:42.47 kernel
ueoffline: engine 3 has outstanding ct-lib
connections and cannot be offlined.
```

If there are tasks that cannot be migrated to another engine within several minutes, the task is killed, and a message similar to this is sent to the error log:

```
00:00000:00000:1999/04/08 15:20:31.26 kernel
Process 57 is killed due to engine offline.
```

Monitoring engine status

Values in the status column of sysengines track the progress of dbcc engine commands:

- online – indicates the engine is online.

- in offline – indicates that dbcc engine(offline) has been run. The engine is still allocated to the server, but is in the process of having its tasks migrated to other engines.
- in destroy – indicates that all tasks have successfully migrated off the engine, and that the server is waiting on the OS level task to deallocate the engine.
- in create – indicates that an engine is in the process of being brought online.

The following command shows the engine number, status, number of tasks affinitied, and the time an engine was brought online:

```
select engine, status, affinitied, starttime
from sysengines
```

```
engine status       affinitied  starttime
------ ----------- ----------- --------------------------
     0 online               12        Mar  5 1999  9:40PM
     1 online                9        Mar  5 1999  9:41PM
     2 online               12        Mar  5 1999  9:41PM
     3 online               14        Mar  5 1999  9:51PM
     4 online                8        Mar  5 1999  9:51PM
     5 in offline           10        Mar  5 1999  9:51PM
```

Logical process management and *dbcc engine(offline)*

If you are using logical process management to bind particular logins or applications to engine groups, use dbcc engine(offline) carefully. If you offline all engines for an engine group:

- The login or application can run on any engine
- An advisory message is sent to the connection logging in to the server

Since engine affinity is assigned when a client logs in, users who are already logged in are not migrated if the engines in the engine group are brought online again with dbcc engine("online").

Monitoring CPU usage

To maintain the correct number of engines, monitor CPU usage with an operating system utility. See the configuration documentation for your platform for the appropriate utility for your operating system.

Managing user connections

If the SMP system supports network affinity migration, each engine handles the network I/O for its connections. During login, Adaptive Server migrates the client connection task from engine 0 to the engine currently servicing the smallest number of connections. The client's tasks run network I/O on that engine (*network affinity*) until the connection is terminated. To determine if your SMP system supports this migration, see the configuration documentation for your platform.

By distributing the network I/O among its engines, Adaptive Server can handle more user connections. The per-process limit on the maximum number of open file descriptors no longer limits the number of connections. Adding more engines linearly increases the maximum number of file descriptors, as stored in the global variable *@@max_connections*.

As you increase the number of engines, Adaptive Server prints the increased *@@max_connections* value to standard output and the error log file after you restart the server. You can query the value as follows:

```
select @@max_connections
```

This number represents the maximum number of file descriptors allowed by the operating system for your process, minus these file descriptors used by Adaptive Server:

- One for each master network listener on engine 0 (one for every "master" line in the interfaces file entry for that Adaptive Server)
- One for each engine's standard output
- One for each engine's error log file
- Two for each engine's network affinity migration channel
- One per engine for configuration
- One per engine for the interfaces file

For example, if Adaptive Server is configured for one engine, and the value of *@@max_connections* equals 1019, adding a second engine increases the value of *@@max_connections* to 2039 (assuming only one master network listener).

You can configure the number of user connections parameter to take advantage of an increased *@@max_connections* limit. However, each time you decrease the number of engines using max online engines, you must also adjust the number of user connections value accordingly. Reconfiguring max online engines or number of user connections is not dynamic, so you must restart the server to change these configuration values. For information about configuring number of user connections, see Chapter 5, "Setting Configuration Parameters."

Configuration parameters that affect SMP systems

Chapter 5, "Setting Configuration Parameters," lists configuration parameters for Adaptive Server. Some of those parameters, such as spinlock ratios, are applicable only to SMP systems.

Configuring spinlock ratio parameters

Spinlock ratio parameters specify the number of internal system resources such as rows in an internal table or cache that are protected by one **spinlock**. A spinlock is a simple locking mechanism that prevents a process from accessing the system resource currently used by another process. All processes trying to access the resource must wait (or "spin") until the lock is released.

Spinlock ratio configuration parameters are meaningful only in multiprocessing systems. An Adaptive Server configured with only one engine has only one spinlock, regardless of the value specified for a spinlock ratio configuration parameter.

Table 20-1 lists system resources protected by spinlocks and the configuration parameters you can use to change the default spinlock ratio.

Table 20-1: Spinlock ratio configuration parameters

Configuration parameter	System resource protected
lock spinlock ratio	Number of lock hash buckets
open index hash spinlock ratio	Index metadata descriptor hash tables
open index spinlock ratio	Index metadata descriptors
open object spinlock ratio	Object metadata descriptors
partition spinlock ratio	Rows in the internal partition caches
user log cache spinlock ratio	User log caches

The value specified for a spinlock ratio parameter defines the ratio of the particular resource to spinlocks, not the number of spinlocks. For example, if 100 is specified for the spinlock ratio, Adaptive Server allocates one spinlock for each 100 resources. The number of spinlocks allocated by Adaptive Server depends on the total number of resources as well as on the ratio specified. The lower the value specified for the spinlock ratio, the higher the number of spinlocks.

Spinlocks are assigned to system resources in one of two ways:

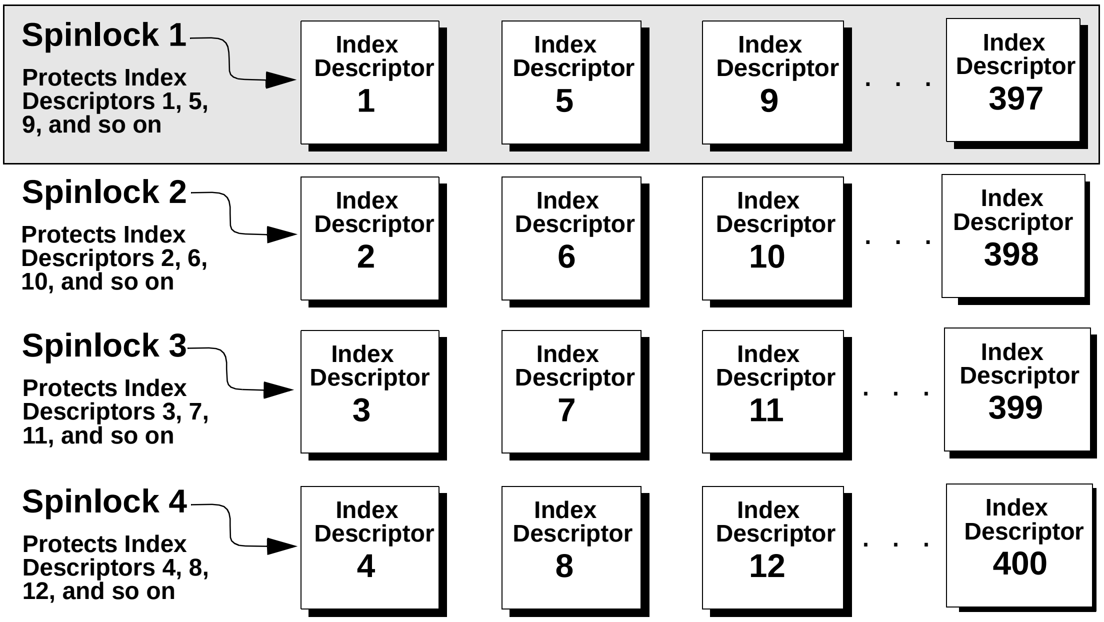
- Round-robin assignment
- Sequential assignment

Round-robin assignment

Metadata cache spinlocks (configured by the open index hash spinlock ratio, open index spinlock ratio, and open object spinlock ratio parameters) use the round-robin assignment method.

Figure 20-2 illustrates one example of the round-robin assignment method and shows the relationship between spinlocks and index metadata descriptors.

Figure 20-2: Relationship between spinlocks and index descriptors



Suppose there are 400 index metadata descriptors, or 400 rows in the index descriptors internal table. You have set the ratio to 100. This means that there will be 4 spinlocks in all: Spinlock 1 protects row 1; Spinlock 2 protects row 2, Spinlock 3 protects row 3, and Spinlock 4 protects row 4. After that, Spinlock 1 protects the next available index descriptor, Index Descriptor 5, until every index descriptor is protected by a spinlock. This round-robin method of descriptor assignment reduces the chances of spinlock contention.

Sequential assignment

Table lock spinlocks, configured by the table lock spinlock ratio parameter, use the sequential assignment method. The default configuration for table lock spinlock ratio is 20, which assigns 20 rows in an internal hash table to each spinlock. The rows are divided up sequentially: the first spinlock protects the first 20 rows, the second spinlock protects the second 20 rows, and so on.

In theory, protecting one resource with one spinlock would provide the least contention for a spinlock and would result in the highest concurrency. In most cases, the default value for these spinlock ratios is probably best for your system. Change the ratio only if there is spinlock contention.

Use sp_sysmon to get a report on spinlock contention. See the *Performance and Tuning Guide* for information on spinlock contention.

CHAPTER 21

Creating and Managing User Databases

This chapter explains how to create and manage user databases.

Topics covered in this chapter include:

Commands for creating and managing user databases

Table 21-1 summarizes the commands for creating, modifying, and dropping user databases and their transaction logs.

Table 21-1: Commands for managing user databases

Command	Task
create database...on *dev_name* or alter database...on *dev_name*	Makes database devices available to a particular Adaptive Server database.. When used without the on *dev_name* clause, these commands allocate space from the default pool of database devices.
dbcc checktable(syslogs)	Reports the size of the log.

Command	Task
sp_logdevice	Specifies a device that will store the log when the current log device becomes full.
sp_helpdb	Reports information about a database's size and devices.
sp_spaceused	Reports a summary of the amount of storage space used by a database.

Permissions for managing user databases

By default, only the System Administrator has create database permission. The System Administrator can grant permission to use the create database command. However, in many installations, the System Administrator maintains a monopoly on create database permission to centralize control of database placement and database device allocation. In these situations, the System Administrator creates new databases on behalf of other users and then transfers ownership to the appropriate user(s).

To create a database and transfer ownership to another user, the System Administrator:

1. Issues the create database command.
2. Switches to the new database with the use *database* command.
3. Executes sp_changedbowner, as described in "Changing database ownership" on page 660.

When a System Administrator grant permission to create databases, the user that receives the permission must also be a valid user of the master database, since all databases are created while using master.

The fact that System Administrators seem to operate outside the protection system serves as a safety precaution. For example, if a Database Owner forgets his or her password or accidentally deletes all entries in sysusers, a System Administrator can repair the damage using the backups or dumps that are made regularly.

Permission alter database or drop database defaults to the Database Owner, and permission is automatically transferred with database ownership. alter database and drop database permission cannot be changed with grant or revoke.

Using the *create database* command

Use create database to create user databases. You must have create database permission, and you must be a valid user of master. Always type use master before you create a new database.

Note Each time you enter the create database command, dump the master database. This makes recovery easier and safer in case master is later damaged. See Chapter 28, "Restoring the System Databases," for more information.

---

create database syntax

The create database syntax is:

```
create database database_name
    [on {default | database_device} [= size]
        [, database_device [= size]]...]
    [log on database_device [ = size ]
        [, database_device [= size]]...]
    [with {override | default_location = "pathname"}]
    [for {load | proxy_update}]
```

A database name must follow the rules for identifiers. You can create only one database at a time.

In its simplest form, create database creates a database on the default database devices listed in master..sysdevices:

```
create database newpubs
```

You can control different characteristics of the new database by using the create database clauses:

- The on clause specifies the names of one or more database devices and the space allocation, in megabytes, for each database device. See "Assigning space and devices to databases" on page 653 for more information.
- The log on clause places the **transaction log** (the syslogs table) on a separate database device with the specified or default size. See "Placing the transaction log on a separate device" on page 655 for more information.

- for load causes Adaptive Server to skip the page-clearing step during database creation. Use this clause if you intend to load a dump into the new database as the next step. See "Using the for load option for database recovery" on page 659 for more information.
- with override allows Adaptive Servers on machines with limited space to maintain their logs on device fragments that are separate from their data. Use this option *only* when you are putting log and data on the same logical device. See "Using the with override option with create database" on page 660 for more information.
- *size* is in the following unit specifiers: 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes).

How *create database* works

When a user with the required permission issues create database, Adaptive Server:

- Verifies that the database name specified is unique.
- Makes sure that the database device names specified are available.
- Finds an unused identification number for the new database.
- Assigns space to the database on the specified database devices and updates master..sysusages to reflect these assignments.
- Inserts a row into sysdatabases.
- Makes a copy of the model database in the new database space, thereby creating the new database's system tables.
- Clears all the remaining pages in the database device. If you are creating a database to load a database dump, for load skips page clearing, which is performed after the load completes).

The new database initially contains a set of system tables with entries that describe the system tables themselves. The new database inherits all the changes you have made to the model database, including:

- The addition of user names.
- The addition of objects.

- The database option settings. Originally, the options are set to "off" in model. If you want all of your databases to inherit particular options, change the options in model with sp_dboption. See Chapter 2, "System and Optional Databases," for more information about model. See Chapter 22, "Setting Database Options," for more information about changing database options.

Adding users to databases

After creating a new database, the System Administrator or Database Owner can manually add users to the database with sp_adduser. This task can be done with the help of the System Security Officer, if new Adaptive Server logins are required. See Chapter 9, "Security Administration," for details on managing Adaptive Server logins and database users.

Assigning space and devices to databases

Adaptive Server allocates storage space to databases when a user enters the create database or alter database command. create database can specify one or more database devices, along with the amount of space on each that is to be allocated to the new database.

Note You can also use the log on clause to place a production database's transaction log on a separate device. See "Placing the transaction log on a separate device" on page 655 for more information.

If you use the default keyword, or if you omit the on clause, Adaptive Server puts the database on one or more of the default database devices specified in master..sysdevices. See "Designating default devices" on page 549 for more information about the default pool of devices.

To specify a size (4MB in the following example) for a database that is to be stored in a default location, use on default = size like this:

```
create database newpubs
on default = “4M”
```

To place the database on specific database devices, give the name(s) of the database device(s) where you want it stored. You can request that a database be stored on more than one database device, with a different amount of space on each. All the database devices named in create database must be listed in sysdevices. In other words, they must have been initialized with disk init. See Chapter 16, “Initializing Database Devices,” for instructions about using disk init.

The following statement creates the newdb database and allocates 3MB on mydata and 2MB on newdata. The database and transaction log are not separated:

```
create database newdb
on mydata = “3M, newdata = “2M”
```

Warning! Unless you are creating a small or noncritical database, always place the log on a separate database device. Follow the instructions under “Placing the transaction log on a separate device” on page 655 to create production databases.

If the amount of space you request on a specific database device is unavailable, Adaptive Server creates the database with as much space as possible on each device and displays a message informing you how much space it has allocated on each database device. This is not considered an error. If there is less than the minimum space necessary for a database on the specified database device, create database fails.

If you create (or alter) a database on a UNIX device file that does not use the dsync setting, Adaptive Server displays an error message in the error log file. For example, if you create the “mydata” device in the previous example does not use dsync, you would see a message similar to:

```
Warning: The database ’newdb’ is using an unsafe virtual device ’mydata’. The
recovery of this database can not be guaranteed.
```

Default database size and devices

If you omit the size parameter in the on clause, Adaptive Server creates the database with a default amount of space. This amount is the larger of the sizes specified by the default database size configuration parameter and the model database.

The size of model and the value of default database size are initially set to the size of the server's logical page. To change the size of model, allocate more space to it with alter database. To change the default database size configuration parameter, use sp_configure. Changing default database size enables you to set the default size for new databases to any size between the server's logical page size and 10,000MB. See "default database size" on page 182 for complete instructions.

If you omit the on clause, the database is created as the default size, as described above. The space is allocated in alphabetical order by database device name, from the default database devices specified in master..sysdevices.

To see the logical names of default database devices, enter:

```
select name
    from sysdevices
    where status & 1 = 1
    order by name
```

sp_helpdevice also displays "default disk" as part of the description of database devices.

Estimating the required space

The size allocation decisions you make are important, because it is difficult to reclaim storage space after it has been assigned. You can always add space; however, you cannot de-allocate space that has been assigned to a database, unless you drop the database first.

You can estimate the size of the tables and indexes for your database by using sp_estspace or by calculating the value. See Chapter 15, "Determining Sizes of Tables and Indexes," in the *Performance and Tuning Guide* for instructions.

Placing the transaction log on a separate device

Use the log on clause of the create database command to place the transaction log (the syslogs table) on a separate database device. Unless you are creating very small, noncritical databases, always place the log on a separate database device. Placing the logs on a separate database device:

- Lets you use dump transaction, rather than dump database, thus saving time and tapes.
- Lets you establish a fixed size for the log to keep it from competing for space with other database activity.
- Creates default free-space threshold monitoring on the log segment and allows you to create additional free-space monitoring on the log and data portions of the database. See Chapter 29, "Managing Free Space with Thresholds," for more information.
- Improves performance.
- Ensures full recovery from hard disk crashes. A special argument to dump transaction lets you dump your transaction log, even when your data device is on a damaged disk.

To specify a size and device for the transaction log, use the log on *device* = *size* clause to create database. The size is in the unit specifiers 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes). For example, the following statement creates the newdb database, allocates 8MB on mydata and 4MB on newdata, and places a 3MB transaction log on a third database device, tranlog:

```
create database newdb
on mydata = “8M”, newdata = “4M”
log on tranlog = “3M”
```

Estimating the transaction log size

The size of the transaction log is determined by:

- The amount of update activity in the associated database
- The frequency of transaction log dumps

This is true whether you perform transaction log dumps manually or use threshold procedures to automate the task. As a general rule, allocate to the log 10 to 25 percent of the space that you allocate to the database.

Inserts, deletes, and updates increase the size of the log. dump transaction decreases its size by writing committed transactions to disk and removing them from the log. Since update statements require logging the “before” and “after” images of a row, applications that update many rows at once should plan on the transaction log being at least twice as large as the number of rows to be updated at the same time, or twice as large as your largest table. Or you can **batch** the updates in smaller groups, performing transaction dumps between the batches.

In databases that have a lot of insert and update activity, logs can grow very quickly. To determine the required log size, periodically check the size of the log. This will also help you choose thresholds for the log and scheduling the timing of transaction log dumps. To check the space used by a database’s transaction log, first use the database. Then enter:

```
dbcc checktable(syslogs)
```

dbcc reports the number of data pages being used by the log. If your log is on a separate device, dbcc checktable also tells you how much space is used and how much is free. Here is sample output for a 2MB log:

```
Checking syslogs
The total number of data pages in this table is 199.
*** NOTICE: Space used on the log segment is 0.39 Mbytes, 19.43%.
*** NOTICE: Space free on the log segment is 1.61 Mbytes, 80.57%.
Table has 1661 data rows.
```

You can also use the following Transact-SQL statement to check on the growth of the log:

```
select count(*) from syslogs
```

Repeat either command periodically to see how fast the log grows.

Default log size and device

If you omit the *size* parameter in the log on clause, Adaptive Server allocates one logical page of storage on the specified log device. If you omit the log on clause entirely, Adaptive Server places the transaction log on the same database device as the data tables.

Moving the transaction log to another device

If you did not use the log on clause to create database, follow the instructions in this section to move your transaction log to another database device.

sp_logdevice moves the transaction log of a database with log and data on the same device to a separate database device. However, the transaction log remains on the original device until the allocated page has been filled and the transaction log has been dumped.

Note If the log and its database share the same device, subsequent use of sp_logdevice affects only future writes to the log; it does not immediately move the first few log pages that were written when the database was created. This creates exposure problems in certain recovery situations, and is not recommended.

The syntax for sp_logdevice is:

sp_logdevice *database_name*, *devname*

The database device you name must be initialized with disk init and must be allocated to the database with create or alter database.

To move the entire transaction log to another device:

1. Execute sp_logdevice, naming the new database device.
2. Execute enough transactions to fill the page that is currently in use. The amount of space you will need to update depends on the size of your logical pages. You can execute dbcc checktable(syslogs) before and after you start updating to determine when a new page is used.
3. Wait for all currently active transactions to finish. You may want to put the database into single-user mode with sp_dboption.
4. Run dump transaction, which removes all the log pages that it writes to disk. As long as there are no active transactions in the part of the log on the old device, all of those pages will be removed. See Chapter 26, “Developing a Backup and Recovery Plan,” for more information.

5 Run sp_helplog to ensure that the complete log is on the new log device.

Note When you move a transaction log, the space no longer used by the transaction log becomes available for data. However, you cannot reduce the amount of space allocated to a device by moving the transaction log.

Transaction logs are discussed in detail in Chapter 26, "Developing a Backup and Recovery Plan."

Using the *for load* option for database recovery

Adaptive Server generally clears all unused pages in the database device when you create a new database. Clearing the pages can take several seconds or several minutes to complete, depending on the size of the database and the speed of your system.

Use the for load option if you are going to use the database for loading from a database dump, either for recovery from media failure or for moving a database from one machine to another. Using for load runs a streamlined version of create database that skips the page-clearing step, and creates a target database that can be used *only* for loading a dump.

If you create a database using for load, you can run only the following commands in the new database before loading a database dump:

- alter database...for load
- drop database
- load database

When you load a database dump, the new database device allocations for the database need to match the usage allocations in the dumped database. See Chapter 27, "Backing Up and Restoring User Databases," for a discussion of duplicating space allocation.

After you load the database dump into the new database, there are no restrictions on the commands you can use.

Using the *with override* option with *create database*

This option allows machines with limited space to maintain their logs on device fragments that are separate from their data. Use this option *only* when you put log and data on the same logical device. Although this is not recommended practice, it may be the only option available on machines with limited storage, especially if you need to get databases back online following a hard disk crash.

You will still be able to dump your transaction log, but if you experience a media failure, you will not be able to access the current log, since it is on the same device as the data. You will be able to recover only to the last transaction log dump, and all transactions between that point and the failure time will be lost.

In the following example, the log and data are on separate fragments of the same logical device:

```
create database littledb
    on diskdev1 = "4M"
    log on diskdev1 = "1M"
    with override
```

The minimum database size you can create is the size of model.

Changing database ownership

A System Administrator might want to create the user databases and give ownership of them to another user after completing some of the initial work. sp_changedbowner changes the ownership of a database. The procedure must be executed by the System Administrator in the database where the ownership will be changed. The syntax is:

sp_changedbowner *loginame* [, true]

The following example makes the user “albert” the owner of the current database and drops the aliases of users who could act as the former “dbo.”

```
sp_changedbowner albert
```

The new owner must already have a login name in Adaptive Server, but he or she cannot be a user of the database or have an alias in the database. You may have to use sp_dropuser or sp_dropalias before you can change a database's ownership. See the Chapter 9, "Security Administration," for more information about changing ownership.

To transfer aliases and their permissions to the new Database Owner, add the second parameter, true.

Note You cannot change ownership of the master database. It is always owned by the "sa" login.

Using the *alter database* command

When your database or transaction log grows to fill all the space allocated with create database, you can use alter database to add storage. You can add space for database objects or the transaction log, or both. You can also use alter database to prepare to load a database from backup.

Permission for alter database defaults to the Database Owner, and is automatically transferred with database ownership. For more information, see "Changing database ownership" on page 660. alter database permission cannot be changed with grant or revoke.

alter database syntax

To extend a database, and to specify where storage space is to be added, use the full alter database syntax:

```
alter database database_name
    [on {default | database_device} [= size]
        [, database_device [= size]]...]
    [log on  {default | database_device} [= size]
        [, database_device [= size]]...]
    [with override]
    [for load]
    [for proxy_update]
```

In its simplest form, alter database adds one logical page from the default database devices. If your database separates log and data, the space you add is used only for data. Use sp_helpdevice to find names of database devices that are in your default list.

To add logical page from a default database device to the newpubs database, enter:

```
alter database newpubs
```

The on and log on clauses operate like the corresponding clauses in create database. You can specify space on a default database device or some other database device, and you can name more than one database device. If you use alter database to extend the master database, you can extend it only on the master device. The minimum increase you can specify is 1MB or one allocation unit, whichever is larger.

To add 3MB to the space allocated for the newpubs database on the database device named pubsdata1, enter:

```
alter database newpubs
on pubsdata1 = "3M"
```

If Adaptive Server cannot allocate the requested size, it allocates as much as it can on each database device, with a minimum allocation of .5MB (256 2K pages) per device. When alter database completes, it prints messages telling you how much space it allocated; for example:

```
Extending database by 1536 pages on disk pubsdata1
```

Check all messages to make sure the requested amount of space was added.

The following command adds 2MB to the space allocated for newpubs on pubsdata1, 3MB on a new device, pubsdata2, and 1MB for the log on tranlog:

```
alter database newpubs
on pubsdata1 = "2M", pubsdata2 =" 3M"
log on tranlog
```

Note Each time you issue the alter database command, dump the master database.

Use with override to create a device fragment containing log space on a device that already contains data or a data fragment on a device already in use for the log. Use this option only when you have no other storage options and when up-to-the-minute recoverability is not critical.

Use for load only after using create database for load to re-create the space allocation of the database being loaded into the new database from a dump. See Chapter 27, “Backing Up and Restoring User Databases,” for a discussion of duplicating space allocation when loading a dump into a new database.

Using the *drop database* command

Use drop database to remove a database from Adaptive Server, thus deleting the database and all the objects in it. This command:

- Frees the storage space allocated for the database
- Deletes references to the database from the system tables in the master database

Only the Database Owner can drop a database. You must be in the master database to drop a database. You cannot drop a database that is open for reading or writing by a user.

The syntax is:

drop database *database_name* [, *database_name*]...

You can drop more than one database in a single statement; for example:

```
drop database newpubs, newdb
```

You must drop all databases from a database device before you can drop the database device itself. The command to drop a device is sp_dropdevice.

After you drop a database, dump the master database to ensure recovery in case master is damaged.

System tables that manage space allocation

To create a database on a database device and allocate a certain amount of space to it, Adaptive Server first makes an entry for the new database in sysdatabases. Then, it checks master..sysdevices to make sure that the device names specified in create database actually exist and are database devices. If you did not specify database devices, or used the default option, Adaptive Server checks master..sysdevices and master..sysusages for free space on all devices that can be used for default storage. It performs this check in alphabetical order by device name.

The storage space from which Adaptive Server gathers the specified amount of storage need not be contiguous. The database storage space can even be drawn from more than one database device. A database is treated as a logical unit, even if it is stored on more than one database device.

Each piece of storage for a database must be at least 1 allocation unit. The first page of each allocation unit is the allocation page. It does not contain database rows like the other pages, but contains an array that shows how the rest of the pages are used.

The *sysusages* table

The database storage information is listed in master..sysusages. Each row in master..sysusages represents a space allocation assigned to a database. Thus, each database has one row in sysusages for each time create database or alter database assigns a fragment of disk space to it.

When you install Adaptive Server, sysusages contains rows for these dbids:

- 1, the master database
- 2, the temporary database, tempdb
- 3, the model database
- 4, the sybsystemprocs database

If you installed auditing, the sybsecurity database will be dbid 5.

As new databases are created or current databases enlarged, new rows are added to sysusages to represent new database allocations.

Here is what sysusages might look like on an Adaptive Server with the five system databases and two user databases (with dbids 6 and 7). Both user databases were created with the log on option. The database with dbid 7 has been given additional storage space with two alter database commands:

```
                                  select dbid, segmap, lstart, size, vstart
                                  from sysusages
dbid   segmap      lstart      size             vstart
------ ----------- ----------- -----------     -------
     1           7           0        1536           4
     2           7           0        1024        2564
     3           7           0        1024        1540
     4           7           0        5120    16777216
     5           7           0       10240    33554432
     6           3           0         512     1777216
     6           4         512         512     3554432
     7           3           0        2048    67108864
     7           4        2048        1024    50331648
     7           3        3072         512    67110912
     7           3        3584        1024    67111424

(10 rows affected)
```

The *segmap* column

The segmap column is a bitmask linked to the segment column in the user database's syssegments table. Since the logsegment in each user database is segment 2, and these user databases have their logs on separate devices, segmap contains 4 (2^2) for the devices named in the log on statement and 3 for the data segment that holds the system segment ($2^0 = 1$) + default segment ($2^1 = 2$).

Some possible values for segments containing data or logs are:

Value	Segment
3	Data only (system and default segments)
4	Log only
7	Data and log

Values higher than 7 indicate user-defined segments. The segmap column is explained more fully in the segments tutorial section in Chapter 23, "Creating and Using Segments."

The *lstart*, *size*, and *vstart* columns

- lstart column – the starting page number in the database of this allocation unit. Each database starts at logical address 0. If additional allocations have been made for a database, as in the case of dbid 7, the lstart column reflects this.
- size column – the number of contiguous pages that are assigned to the same database. The ending logical address of this portion of the database can be determined by adding the values in lstart and size.
- vstart column – the address where the piece assigned to this database begins. The upper 4 bits store the virtual device number (vdevno), and the lower 4 bits store the virtual block number. (To obtain the virtual device number, divide sysusages.vstart or sysdevices.low by 16,777,216, which is 2^{24}.) The value in vstart identifies which database device contains the page number of the database, because it falls between the values in the low and high columns of sysdevices for the database device in question.

Getting information about database storage

This section explains how to determine which database devices are currently allocated to databases and how much space each database uses.

Database device names and options

To find the names of the database devices on which a particular database resides, use sp_helpdb with the database name:

```
                           sp_helpdb pubs2
name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     2.0 MB     sa           5 Aug 25, 1997   no options set

device_fragments    size         usage           free kbytes
------------------- ------------ --------------- -----------
pubdev              2.0 MB       data and log            288

device                 segment
---------------------- ----------------------
```

```
pubdev                 default
pubdev                 logsegment
pubdev                 system
```

sp_helpdb reports on the size and usage of the devices used by the named database. The status column lists the database options. These options are described in Chapter 22, “Setting Database Options.”

If you are using the named database, sp_helpdb also reports on the segments in the database and the devices named by the segments. See Chapter 23, “Creating and Using Segments,” for more information.

When you use sp_helpdb without arguments, it reports information about all databases in Adaptive Server:

```
                                    sp_helpdb
name           db_size  owner dbid created      status
-------------- -------- ----- ---- ------------ -------------------
master           3.0 MB sa       1 Jan 01, 1900 no options set
model            2.0 MB sa       3 Jan 01, 1900 no options set
mydata           4.0 MB sa       7 Aug 25, 1997 no options set
pubs2            2.0 MB sa       6 Aug 23, 1997 no options set
sybsecurity     20.0 MB sa       5 Aug 18, 1997 no options set
sybsystemprocs  10.0 MB sa       4 Aug 18, 1997 trunc log on chkpt
tempdb           2.0 MB sa       2 Aug 18, 1997 select into/
                                                bulkcopy/pllsort
```

Checking the amount of space used

sp_spaceused provides:

- A summary of space used in the database
- A summary of space used by a table and its indexes and text/image storage
- A summary of space used by a table, with separate information on indexes and text/image storage.

Checking space used in a database

To get a summary of the amount of storage space used by a database, execute sp_spaceused in the database:

```
sp_spaceused
database_name                    database_size
```

```
------------------------------ ------------
pubs2                          2.0 MB

reserved      data          index_size      unused
------------- ------------- --------------- -------
-
1720 KB       536 KB        344 KB          840 KB
```

Table 21-2 describes the columns in the report.

Table 21-2: Columns in sp_spaceused output

Column	Description
database_name	The name of the database being examined.
database_size	The amount of space allocated to the database by create database or alter database.
reserved	The amount of space that has been allocated to all the tables and indexes created in the database. (Space is allocated to database objects inside a database in increments of 1 extent, or 8 pages, at a time.)
data, index_size	The amount of space used by data and indexes.
unused	The amount of space that has been reserved but not yet used by existing tables and indexes.

The sum of the values in the unused, index_size, and data columns should equal the figure in the reserved column. Subtract reserved from database_size to get the amount of unreserved space. This space is available for new or existing objects that grow beyond the space that has been reserved for them.

By running sp_spaceused regularly, you can monitor the amount of database space available. For example, if the reserved value is close to the database_size value, you are running out of space for new objects. If the unused value is also small, you are running out of space for additional data as well.

Checking summary information for a table

You can also use sp_spaceused with a table name as its parameter:

```
sp_spaceused titles
name   rowtotal reserved  data    index_size unused
------ -------- --------- ------- ---------- -----
titles 18       48 KB     6 KB    4 KB       38 KB
```

The rowtotal column may be different than the results of running select count(*) on the table. This is because sp_spaceused computes the value with the built-in function rowcnt. That function uses values that are stored in the allocation pages. These values are not updated regularly, however, so they can be very different for tables with a lot of activity. update statistics, dbcc checktable, and dbcc checkdb update the rows-per-page estimate, so rowtotal will be most accurate after you have run one of these commands has been run.

You should run sp_spaceused regularly on syslogs, since the transaction log can grow rapidly if there are frequent database modifications. This is particularly a problem if the transaction log is not on a separate device—in which case, it competes with the rest of the database for space.

Checking information for a table and its indexes

To see information on the space used by individual indexes, enter:

```
                                 sp_spaceused titles, 1
index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
titleidind           2 KB       32 KB      24 KB
titleind             2 KB       16 KB      14 KB

name       rowtotal   reserved   data   index_size unused
---------- --------   ---------  ------- ---------- ----------
titles          18      46 KB     6 KB      4 KB      36 KB
```

Space taken up by the text/image page storage is reported separately from the space used by the table. The object name for text/image storage is always "t" plus the table name:

```
                                 sp_spaceused blurbs,1
index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
blurbs               0 KB       14 KB      12 KB
tblurbs              14 KB      16 KB      2 KB

name       rowtotal reserved    data    index_size unused
---------- -------- ----------- ------- ---------- ----------
blurbs            6       30 KB    2 KB      14 KB      14 KB
```

Querying system table for space usage information

You may want to write some of your own queries for additional information about physical storage. For example, to determine the total number of 2K blocks of storage space that exist on Adaptive Server, you can query sysdevices:

```
select sum(high - low)
from sysdevices
where status in (2, 3)
-------------------
               7168
```

A 2 in the status column represents a physical device; a 3 represents a physical device that is also a default device.

CHAPTER 22 Setting Database Options

This chapter describes how to use database options.

Topics covered in this chapter include:

What are database options?

Database options control:

- The behavior of transactions
- Defaults for table columns
- Restrictions to user access
- Performance of recovery and bcp operations
- Log behavior

The System Administrator and the Database Owner can use database options to configure the settings for an entire database. Database options differ from sp_configure parameters, which affect the entire server, and set options, which affect only the current session or stored procedure.

Using the *sp_dboption* procedure

Use sp_dboption to change settings for an entire database. The options remain in effect until they are changed. sp_dboption:

- Displays a complete list of the database options when it is used without a parameter
- Changes a database option when used with parameters

You can change options for user databases only. You cannot change options for the master database. To change a database option in a user database (or to display a list of the database options), execute sp_dboption while using the master database.

The syntax is:

sp_dboption [*dbname*, *optname*, {true | false}]

To make an option or options take effect for every new database, change the option in the model database.

Database option descriptions

All users with access to the master database can execute sp_dboption with no parameters to display a list of the database options. The report from sp_dboption looks like this:

```
sp_dboption
Settable database options.
--------------------
abort tran on log full
allow nulls by default
auto identity
dbo use only
ddl in tran
identity in nonunique index
no chkpt on recovery
no free space acctg
read only
select into/bulkcopy/pllsort
single user
trunc log on chkpt
trunc. log on chkpt.
unique auto_identity index
```

For a report on which options have been set in a particular database, execute sp_helpdb in that database.

The following sections describe each database option in detail.

abort tran on log full

abort tran on log full determines the fate of a transaction that is running when the last-chance threshold is crossed. The default value is false, meaning that the transaction is suspended and is awakened only when space has been freed. If you change the setting to true, all user queries that need to write to the transaction log are killed until space in the log has been freed.

allow nulls by default

Setting allow nulls by default to true changes the default null type of a column from not null to null, in compliance with the SQL standard. The Transact-SQL default value for a column is not null, meaning that null values are not allowed in a column unless null is specified in the create table or alter table column definition.

auto identity

While the auto identity option is true, a 10-digit IDENTITY column is defined in each new table that is created without specifying either a primary key, a unique constraint, or an IDENTITY column. This IDENTITY column is only created when you issue a create table command, not when you issue a select into. The column is not visible when you select all columns with the select * statement. To retrieve it, you must explicitly mention the column name, SYB_IDENTITY_COL, in the select list.

To set the precision of the automatic IDENTITY column, use the size of auto identity configuration parameter.

Though you can set auto identity to true in tempdb, it is not recognized or used, and temporary tables created there do not automatically include an IDENTITY column.

dbo use only

While dbo use only is set to true (on), only the Database Owner can use the database.

ddl in tran

Setting ddl in tran to true allows these commands to be used inside a user-defined transaction:

- alter table (clauses other than partition and unpartition are allowed)
- create default
- create index
- create procedure
- create rule
- create schema
- create table
- create trigger
- create view
- drop default
- drop index
- drop procedure
- drop rule
- drop table
- drop trigger
- drop view
- grant
- revoke

Data definition statements lock system tables for the duration of a transaction, which can result in performance problems. Use them only in short transactions.

These commands cannot be used in a user-defined transaction under any circumstances:

- alter database
- alter table...partition
- alter table...unpartition
- create database

- disk init
- dump database
- dump transaction
- drop database
- load transaction
- load database
- select into
- truncate table
- update statistics

identity in nonunique index

identity in nonunique index automatically includes an IDENTITY column in a table's index keys so that all indexes created on the table are unique. This database option makes logically nonunique indexes internally unique and allows those indexes to be used to process updatable cursors and isolation level 0 reads.

The table must already have an IDENTITY column for the identity in nonunique index option to work either from a create table statement or from setting the auto identity database option to true before creating the table.

Use identity in nonunique index if you plan to use cursors and isolation level 0 reads on tables that have nonunique indexes. A unique index ensures that the cursor is positioned at the correct row the next time a fetch is performed on that cursor.

Do not confuse the identity in nonunique index option with unique auto_identity index, which is used to add an IDENTITY column with a unique, nonclustered index to new tables.

no chkpt on recovery

no chkpt on recovery is set to true (on) when an up-to-date copy of a database is kept. In these situations, there is a "primary" database and a "secondary" database. Initially, the primary database is dumped and loaded into the secondary database. Then, at intervals, the transaction log of the primary database is dumped and loaded into the secondary database.

If this option is set to false (off)—the default—a checkpoint record is added to the database after it is recovered by restarting Adaptive Server. This checkpoint, which ensures that the recovery mechanism is not re-run unnecessarily, changes the sequence number of the database. If the sequence number of the secondary database has been changed, a subsequent dump of the transaction log from the primary database cannot be loaded into it.

Turning this option on for the secondary database causes it to not get a checkpoint from the recovery process so that subsequent transaction log dumps from the primary database can be loaded into it.

no free space acctg

no free space acctg suppresses free-space accounting and execution of threshold actions for the non-log segments. This speeds recovery time because the free-space counts will not be recomputed for those segments. It disables updating the rows-per-page value stored for each table, so system procedures that estimate space usage may report inaccurate values.

read only

read only means that users can retrieve data from the database, but cannot modify anything.

select into/bulkcopy/pllsort

select into/bulkcopy/pllsort must be set to on to perform operations that do not keep a complete record of the transaction in the log, which include:

- Using the writetext utility.
- Doing a select into a permanent table.

- Doing a "fast" **bulk copy** with bcp. By default, fast bcp is used on tables that do not have indexes.
- Performing a parallel sort.

Adaptive Server performs minimal logging for these commands, recording only page allocations and deallocations, but not the actual changes made to the data pages.

You do not have to set select into/bulkcopy/pllsort on to select into a temporary table, since tempdb is never recovered. Additionally, you do not need to set the option to run bcp on a table that has indexes, because inserts are logged.

After you have run select into or performed a bulk copy in a database, you will not be able to perform a regular transaction log dump. After you have made minimally logged changes to your database, you must perform a dump database, since changes are not recoverable from transaction logs.

Setting select into/bulkcopy/pllsort does not block log dumping, but making minimally logged changes to data does block the use of a regular dump transaction. However, you can still use dump transaction...with no_log and dump transaction...with truncate_only.

By default, select into/bulkcopy/pllsort is turned off in newly created databases. To change the default, turn this option on in the model database.

single user

When single user is set to true, only one user at a time can access the database. You cannot set single user to true in tempdb.

trunc log on chkpt

When trunc log on chkpt is true (on), the transaction log is truncated (committed transactions are removed) when the checkpoint checking process occurs (usually more than once per minute), if 50 or more rows have been written to the log. The log is *not* truncated if less than 50 rows were written to the log, or if the Database Owner runs the checkpoint command manually.

You may want to turn this option on while doing development work during which backups of the transaction log are not needed. If this option is off (the default), and the transaction log is never dumped, the transaction log continues to grow, and you may run out of space in your database.

When trunc log on chkpt is on, you cannot dump the transaction log because changes to your data are not recoverable from transaction log dumps. Use dump database instead.

By default, the trunc log on chkpt option is off in newly created databases. To change the default, turn this option on in the model database.

Warning! If you set trunc log on chkpt on in model, and you need to load a set of database and transaction logs into a newly created database, be sure to turn the option off in the new database.

unique auto_identity index

When the unique auto_identity index option is set to true, it adds an IDENTITY column with a unique, nonclustered index to new tables. By default, the IDENTITY column is a 10-digit numeric datatype, but you can change this default with the size of auto identity column configuration parameter.

Though you can set unique auto_identity index to true in tempdb, it is not recognized or used, and temporary tables created there do not automatically include an IDENTITY column with a unique index.

The unique auto_identity index option provides a mechanism for creating tables that have an automatic IDENTITY column with a unique index that can be used with updatable cursors. The unique index on the table ensures that the cursor is positioned at the correct row after a fetch. (If you are using isolation level 0 reads and need to make logically nonunique indexes internally unique so that they can process updatable cursors, use the identity in nonunique index option.)

In some cases, the unique auto_identity index option can avoid the Halloween Problem for the following reasons:

- Users cannot update an IDENTITY column; hence, it cannot be used in the cursor update.

- The IDENTITY column is automatically created with a unique, nonclustered index so that it can be used for the updatable cursor scan.

For more information about the Halloween Problem, IDENTITY columns, and cursors, see the *Transact-SQL User's Guide*.

Do not confuse the unique auto_identity index option with the identity in nonunique index option, which is used to make all indexes in a table unique by including an IDENTITY column in the table's index keys.

Changing database options

Only a System Administrator or the Database Owner can change a user's database options by executing sp_dboption. Users aliased to the Database Owner cannot change database options with sp_dboption.

You must be using the master database to execute sp_dboption. Then, for the change to take effect, you must issue the checkpoint command while using the database for which the option was changed.

Remember that you cannot change any master database options.

To change pubs2 to read only:

```
use master
sp_dboption pubs2, "read only", true
```

Then, run the checkpoint command in the database that was changed:

```
use pubs2
checkpoint
```

For the *optname* parameter of sp_dboption, Adaptive Server understands any unique string that is part of the option name. To set the trunc log on chkpt option:

```
use master
sp_dboption pubs2, trunc, true
```

If you enter an ambiguous value for *optname*, an error message is displayed. For example, two of the database options are dbo use only and read only. Using "only" for the *optname* parameter generates a message because it matches both names. The complete names that match the string supplied are printed out so that you can see how to make the *optname* more specific.

You can turn on more than one database option at a time. You cannot change database options inside a user-defined transaction.

Viewing the options on a database

Use sp_helpdb to determine the options that are set for a particular database. sp_helpdb lists each active option in the "status" column of its output.

The following example shows that the read only option is turned on in mydb:

```
                           sp_helpdb mydb
name  db_size owner dbid created    status
-----  -------  -----  ----  ------------  ----------------------
mydb   2.0 MB    sa     5    Mar 05, 1999 read only
device_fragments      size    usage           free kbytes
-----------------     ------  ------------  -------------
master                2.0 MB  data and log            576
device                         segment
------------------------------ ------------------------------
master                         default
master                         logsegment
master                         system
name    attribute_class attribute  int_value char_value
         comments
------- --------------- ---------- --------- --------------------------
         -----------
pubs2   buffer manager  cache name      NULL cache for database mydb
         NULL
```

To display a summary of the options for all databases, use sp_helpdb without specifying a database:

```
                           sp_helpdb
name                 db_size         owner         dbid
   created        status
-------------------  --------------  -------------  -----
   -------------  ---------------------------
mydb                 2.0 MB          sa            5
   May 10, 1997   read only
master               3.0 MB          sa            1
   Jan 01, 1997   no options set
model                2.0 MB          sa            3
```

```
   Jan 01, 1997   no options set
sybsystemprocs      2.0 MB           sa            4
   Mar 31, 1995   trunc log on chkpt
tempdb              2.0 MB           sa            2
   May 04, 1998 select into/bulkcopy/pllsort
```

CHAPTER 23 Creating and Using Segments

This chapter introduces the system procedures and commands for using segments in databases.

Topics covered in this chapter include:

See also Chapter 5, "Controlling Physical Data Placement," in the *Performance and Tuning Guide* for information about how segments can improve system performance.

What is a segment?

A segment is a label that points to one or more database devices. Segment names are used in create table and create index commands to place tables or indexes on specific database devices. Using segments can improve Adaptive Server performance and give the System Administrator or Database Owner increased control over the placement, size, and space usage of database objects.

You create segments within a database to describe the database devices that are allocated to the database. Each Adaptive Server database can contain up to 32 segments, including the system-defined segments (see “System-defined segments” on page 684). Before assigning segment names, you must initialize the database devices with disk init and then make them available to the database with create database or alter database.

System-defined segments

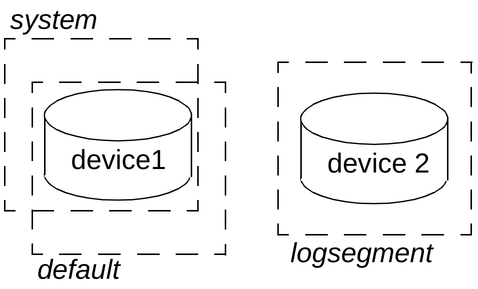
When you first create a database, Adaptive Server creates three segments in the database, as described in Table 23-1.

Table 23-1: System-defined segments

Segment	Function
system	Stores the database’s system tables
logsegment	Stores the database’s transaction log
default	Stores all other database objects—unless you create additional segments and store tables or indexes on the new segments by using create table...on *segment_name* or create index...on *segment_name*

If you create a database on a single database device, the system, default, and logsegment segments label the same device. If you use the log on clause to place the transaction log on a separate device, the segments resemble those shown in Figure 23-1.

Figure 23-1: System-defined segments



Although you can add and drop user-defined segments, you cannot drop the default, system, or log segments from a database. A database must have at least one default, system-defined, and log segment.

Commands and procedures for managing segments

Table 23-2 summarizes the commands and system procedures for managing segments.

Table 23-2: Commands and procedures for managing segments

Command or procedure	Function
sp_addsegment	Defines a segment in a database.
create table and create index	Creates a database object on a segment.
sp_dropsegment	Removes a segment from a database or removes a single device from the scope of a segment.
sp_extendsegment	Adds devices to an existing segment.
sp_placeobject	Assigns future space allocations for a table or an index to a specific segment.
sp_helpsegment	Displays the segment allocation for a database or data on a particular segment.
sp_helpdb	Displays the segments on each database device. See Chapter 21, “Creating and Managing User Databases,” for examples.
sp_help	Displays information about a table, including the segment where the table resides.
sp_helpindex	Displays information about a table’s indexes, including the segments where the indexes reside.

Why use segments?

When you add a new device to a database, Adaptive Server places the new device in a default pool of space (the database’s default and system segments). This increases the total space available to the database, but it does not determine which objects will occupy that new space. Any table or index might grow to fill the entire pool of space, leaving critical tables with no room for expansion. It is also possible for several heavily used tables and indexes to be placed on a single physical device in the default pool of space, resulting in poor I/O performance.

When you create an object on a segment, the object can use all the database devices that are available in the segment, but no other devices. You can use segments to control the space that is available to individual objects.

The following sections describe how to use segments to control disk space usage and to improve performance. "Moving a table to another device" on page 689 explains how to move a table from one device to another using segments and clustered indexes.

Controlling space usage

If you assign noncritical objects to a segment, those objects cannot grow beyond the space available in the segment's devices. Conversely, if you assign a critical table to a segment, and the segment's devices are not available to other segments, no other objects will compete with that table for space.

When the devices in a segment become full, you can extend the segment to include additional devices or device fragments as needed. Segments also allow you to use thresholds to warn you when space becomes low on a particular database segment.

If you create additional segments for data, you can create new threshold procedures for each segment. See Chapter 29, "Managing Free Space with Thresholds," for more information on thresholds.

Improving performance

In a large, multidatabase and/or multidrive Adaptive Server environment, you can enhance system performance by paying careful attention to the allocation of space to databases and the placement of database objects on physical devices. Ideally, each database has exclusive use of database devices, that is, it does not share a physical disk with another database. In most cases, you can improve performance by placing heavily used database objects on dedicated physical disks or by "splitting" large tables across several physical disks.

The following sections describe these ways to improve performance. The *Performance and Tuning Guide* also offers more information about how segments can improve performance.

Separating tables, indexes, and logs

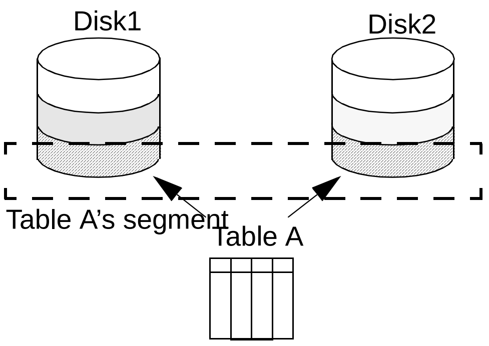
Generally, placing a table on one physical device, its nonclustered indexes on a second physical device, and the transaction log on a third physical device can speed performance. Using separate physical devices (disk controllers) reduces the time required to read or write to the disk. If you cannot devote entire devices in this way, at least restrict all nonclustered indexes to a dedicated physical device.

The log on extension to create database (or sp_logdevice) places the transaction log on a separate physical disk. Use segments to place tables and indexes on specific physical devices. See "Assigning database objects to segments" on page 692 for information about placing tables and indexes on segments.

Splitting tables

You can split a large, heavily used table across devices on separate disk controllers to improve the overall read performance of a table. When a large table exists on multiple devices, it is more likely that small, simultaneous reads will take place on different disks. Figure 23-2 shows a table that is split across the two devices in its segment.

Figure 23-2: Partitioning a table across physical devices



You can split a table across devices using one of three different methods, each of which requires the use of segments:

- Use table partitioning.
- If the table has a clustered index, use partial loading.

- If the table contains text or image datatypes, separate the text chain from other data.

Partitioning tables

Partitioning a table creates multiple page chains for the table and distributes those page chains over all the devices in the table's segment (see Figure 23-2). Partitioning a table increases both insert and read performance, since multiple page chains are available for insertions.

Before you can partition a table, you must create the table on a segment that contains the desired number of devices. The remainder of this chapter describes how to create and modify segments. See Chapter 5, "Controlling Physical Data Placement," of the *Performance and Tuning Guide* for information about partitioning tables using alter table.

Partial loading

To split a table with a clustered index, use sp_placeobject with multiple load commands to load different parts of the table onto different segments. This method can be difficult to execute and maintain, but it provides a way to split tables and their clustered indexes across physical devices. See "Placing existing objects on segments" on page 694 for more information and syntax.

Separating text and image columns

Adaptive Server stores the data for text and image columns on a separate chain of data pages. By default, this text chain is placed on the same segment as the table's other data. Since reading a text column requires a read operation for the text pointer in the base table and an additional read operation on the text page in the separate text chain, placing the text chain and base table data on a separate physical device can improve performance. See "Placing text pages on a separate device" on page 696 for more information and syntax.

Moving a table to another device

You can also use segments to move a table from one device to another using the create clustered index command. Clustered indexes, where the bottom or *leaf level* of the index contains the actual data, are on the same segment as the table. Therefore, you can completely move a table by dropping its clustered index (if one exists), and creating or re-creating a clustered index on the desired segment. See "Creating clustered indexes on segments" on page 697 for more information and syntax.

Creating segments

To create a segment in a database:

- Initialize the physical device with disk init.
- Make the database device available to the database by using the on clause to create database or alter database. This automatically adds the new device to the database's default and system segments.

Once the database device exists and is available to the database, define the segment in the database with the stored procedure sp_addsegment. The syntax is:

```
sp_addsegment segname, dbname, devname
```

where:

- segname is any valid identifier. Give segments names that identify what they are used for, and use extensions like "_seg."
- dbname is the name of the database where the segment will be created.
- devname is the name of the database device—the name used in disk init and the create and alter database statements.

This statement creates the segment seg_mydisk1 on the database device mydisk1:

```
sp_addsegment seg_mydisk1, mydata, mydisk1
```

Changing the scope of segments

When you use segments, you also need to manage their scope – the number of database devices to which each segment points. You can:

- Extend the scope of a segment by making it point to an additional device or devices, or
- Reduce the scope of a segment by making it point to fewer devices.

Extending the scope of segments

You may need to extend a segment if the database object or objects assigned to the segment run out of space. sp_extendsegment extends the size of a segment by including additional database devices as part of an existing segment. The syntax is:

sp_extendsegment *segname*, *dbname*, *devname*

Before you can extend a segment:

- The database device must be listed in sysdevices,
- The database device must be available in the desired database, and
- The segment name must exist in the current database.

The following example adds the database device pubs_dev2 to an existing segment named bigseg:

```
sp_extendsegment bigseg, pubs2, pubs_dev2
```

To extend the default segment in your database, you must place the word "default" in quotes:

```
sp_extendsegment "default", mydata, newdevice
```

Automatically extending the scope of a segment

If you use alter database to add space on a database device that is new to the database, the system and default segments are extended to include the new space. Thus, the scope of the system and default segments is extended each time you add a new device to the database.

If you use alter database to assign additional space on an existing database device, all the segments mapped to the existing device are extended to include the new device fragment. For example, assume that you initialized a 4MB device named newdev, allocated 2MB of the device to mydata, and assigned the 2MB to the testseg segment:

```
alter database mydata on newdev = "2M"
sp_addsegment testseg, mydata, newdev
```

If you alter mydata later to use the remaining space on newdev, the remaining space fragment is automatically mapped to the testseg segment:

```
alter database mydata on newdev = "2M"
```

See "A segment tutorial" on page 701 for more examples about how Adaptive Server assigns new device fragments to segments.

Reducing the scope of a segment

You may need to reduce the scope of a segment if it includes database devices that you want to reserve exclusively for other segments. For example, if you add a new database device that is to be used exclusively for one table, you will want to reduce the scope of the default and system segments so that they no longer point to the new device.

Use sp_dropsegment to drop a single database device from a segment, reducing the segment's scope:

sp_dropsegment *segname*, *dbname*, *device*

With three arguments, sp_dropsegment drops only the given device from the scope of devices spanned by the segment. You can also use sp_dropsegment to remove an entire segment from the database, as described under "Dropping segments" on page 697.

The following example removes the database device pubs_dev2 from the scope of bigseg:

```
sp_dropsegment bigseg, pubs2, pubs_dev2
```

Assigning database objects to segments

This section explains how to assign new or existing database objects to user-defined segments to:

- Restrict new objects to one or more database devices
- Place a table and its index on separate devices to improve performance
- Split an existing object over multiple database devices

Creating new objects on segments

To place a new object on a segment, first create the new segment. You may also want to change the scope of this segment (or other segments) so that it points only to the desired database devices. Remember that when you add a new database device to a database, it is automatically added to the scope of the default and system segments.

After you have defined the segment in the current database, use create table or create index with the optional on *segment_name* clause to create the object on the segment. The syntax is:

```
create table table_name (col_name datatype ... )
    [on segment_name]
```

```
create [ clustered | nonclustered ] index index_name
    on table_name(col_name)
    [on segment_name]
```

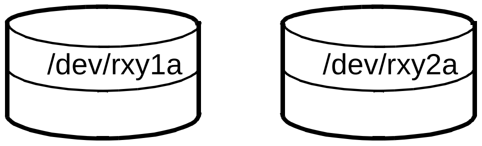
Note Clustered indexes, where the bottom leaf, or leaf level, of the index contains the actual data, are by definition on the same segment as the table. See "Creating clustered indexes on segments" on page 697.

Example: creating a table and index on separate segments

Figure 23-3 summarizes the sequence of Transact-SQL commands used to create tables and indexes on specific physical disks on a server using 2K logical page size.

Figure 23-3: Creating objects on specific devices using segments

Physical devices



Select physical devices to be used by Adaptive Server.

(1) **use *master***

Start in *master* database.

(2)

```
disk init
name = "mydisk1",
physname =
"/dev/rxy1a",
vdevno = 7,
size = 2048
```

```
disk init
name = "mydisk2",
physname =
"/dev/rxy2a",
vdevno = 8,
size = 1024
```

Map Adaptive Server database device name to physical device with **disk init.**

(3)

```
alter database mydata
on mydisk1 = 4, mydisk2 = 2
```

Add the devices *mydisk1* and *mydisk2* to *mydata*.

(4) **use mydata**

Change to *mydata* database.

(5)

```
sp_addsegment seg_mydisk1, mydata, mydisk1
sp_addsegment seg_mydisk2, mydata, mydisk2
```

Map segment names to database device names.

(6)

```
sp_dropsegment "default", mydata, mydisk1
sp_dropsegment system, mydata, mydisk1
sp_dropsegment "default", mydata, mydisk2
sp_dropsegment system, mydata, mydisk2
```

Drop devices from the scope of *system* and *default*.

(7)

```
create table authors (au_id...) on seg_mydisk1
create nonclustered index au_index on authors
(au_id)
```

Create table on one segment, and create its index on the other segment.

1 Start by using the master database.

2 Initialize the physical disks.

3 Allocate the new database devices to a database.

4 Change to the mydata database using the use *database* command.

5 Create two new segments, each of which points to one of the new devices.

6 Reduce the scope of the default and system segments so that they do not point to the new devices.

7 Create the objects, giving the new segment names.

Placing existing objects on segments

sp_placeobject does not remove an object from its allocated segment. However, it causes all further disk allocation for that object to occur on the new segment it specifies. The syntax is:

sp_placeobject *segname*, *objname*

The following command causes all further disk allocation for the mytab table to take place on bigseg:

```
sp_placeobject bigseg, mytab
```

sp_placeobject does not move an object from one database device to another. Whatever pages have been allocated on the first device remain allocated; whatever data was written to the first device remains on the device. sp_placeobject affects only future space allocations.

Note To completely move a table, you can drop its clustered index (if one exists), and create or re-create a clustered index on the desired segment. To completely move a nonclustered index, drop the index and re-create it on the new segment. See "Creating clustered indexes on segments" on page 697 for instructions on moving a table.

After you have used sp_placeobject, executing dbcc checkalloc causes the following message to appear for each object that is split across segments:

```
Extent not within segment: Object object_name, indid
index_id includes extents on allocation page
page_number which is not in segment segment_name.
```

You can ignore this message.

Example: splitting a table and its clustered index across physical devices

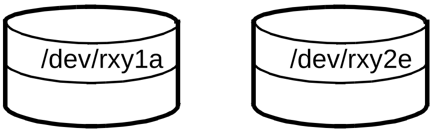
Performance can be improved for high-volume, multiuser applications when large tables are split across segments that are located on separate disk controllers.

The order of steps is quite important at certain stages. In particular, you must create the clustered index before you place the table is placed on the second segment.

Figure 23-4 summarizes the process of splitting a table across two segments on a server using 2K logical page size:

Figure 23-4: Splitting a large table across two segments

Physical devices



Select physical devices to be used by Adaptive Server.

1	**use master**	Start in *master* database.
2	**disk init** **name = "mydisk1",** **physname =** **"/dev/rxy1a",** **vdevno = 7,** **size = 2048** **disk init** **name = "mydisk2",** **physname =** **"/dev/rxy2e",** **vdevno = 8,** **size = 2048**	Map Adaptive Server database device name to physical device with **disk init**.
3	**alter database mydata** **on mydisk1 = 4, mydisk2 = 4**	Add the devices *mydisk1* and *mydisk2* to *mydata*.
4	**use mydata**	Change to *mydata* database.
5	**sp_addsegment seg_mydisk1, mydata, mydisk1** **sp_addsegment seg_mydisk2, mydata, mydisk2** **sp_addsegment seg_bothdisks, mydata, mydisk1** **sp_extendsegment seg_bothdisks, mydata, mydisk2**	Add a segment on *mydisk1* and another on *mydisk2*. Create a third segment, and extend it to span both disks.
6	**sp_dropsegment "default", mydata, mydisk1** **sp_dropsegment system, mydata, mydisk1** **sp_dropsegment "default", mydata, mydisk2** **sp_dropsegment system, mydata, mydisk2**	Drop devices from the scope of *system* and *default*.
7	**create table authors (au_id etc.) on seg_mydisk1** **create clustered index au_ind on authors (au_id)** **on seg_mydisk1**	Create the table and clustered index on the segment.
8	[use **bcp** to load half of the rows]	Load half of the rows.
9	**sp_placeobject segmydisk2, authors**	Place the object on the second segment.
10	[use **bcp** to load the rest of the rows]	Load the rest of the rows.
11		Place the table on the segment that spans both disks.

1. Begin by using the master database.
2. Initialize the devices with disk init.
3. Assign both devices to the mydata database with alter database.
4. Change to the mydata database by entering the use *database* command.
5. Create three segments. The first two should each point to one of the new devices. Extend the scope of the third segment so that it labels both devices.
6. Drop the system and default segments from both devices.
7. Create the table and its clustered index on the first segment.
8. Load half of the table's data onto the first segment.
9. Use sp_placeobject to cause all further allocations of disk space to occur on the second segment.
10. Load the remaining data onto the second segment.
11. Use sp_placeobject again to place the table on the segment that spans both devices.

The balance of disk allocation may change over time if the table is updated frequently. To guarantee that the speed advantages are maintained, you may need to drop and re-create the table at some point.

Placing text pages on a separate device

When you create a table with text or image columns, the data is stored on a separate chain of text pages. A table with text or image columns has an additional entry in sysindexes for the text chain, with the name column set to the name of the table preceded by the letter "t" and an indid of 255. You can use sp_placeobject to store the text chain on a separate device, giving both the table name and the name of the text chain from sysindexes:

```
sp_placeobject textseg, "mytab.tmytab"
```

Note By default, a chain of text pages is placed on the same segment as its table. After you execute sp_placeobject, pages that were previously written on the old device remain allocated, but all new allocations take place on the new segment.

Creating clustered indexes on segments

The bottom, or leaf level, of a clustered index contains the data. Therefore, a table and its clustered index are on the same segment. If you create a table on one segment and its clustered index on a different segment, the table will migrate to the segment where you created the clustered index. This provides a quick and easy way to move a table to other devices in your database.

The syntax for creating a clustered index on a segment is:

```
create [unique] clustered index index_name
    on [[database.]owner.]table_name (column_name
        [, column_name]...)
    [with {fillfactor = x, ignore_dup_key, sorted_data,
        [ignore_dup_row | allow_dup_row]}]
    on segment_name
```

See "Segments and clustered indexes" on page 706 for an example of this command.

Dropping segments

When you use sp_dropsegment with only a segment name and the database name, the named segment is dropped from the database. However, you cannot drop a segment as long as database objects are still assigned to it. You must assign the objects to another segment or drop the objects first and then drop the segment.

The syntax for dropping a segment is:

```
sp_dropsegment segname, dbname
```

You cannot completely drop the default, system, or log segment from a database. A database must have at least one default, system, and log segment. You can, however, reduce the scope of these segments–see "Reducing the scope of a segment" on page 691.

Note Dropping a segment removes its name from the list of segments in the database, but it does not remove database devices from the allocation for that database, nor does it remove objects from devices.
If you drop all segments from a database device, the space is still allocated to the database but cannot be used for database objects. dbcc checkcatalog reports "Missing segment in Sysusages segmap." To make a device available to a database, use sp_extendsegment to map the device to the database's default segment:

```
sp_extendsegment "default", dbname, devname
```

Getting information about segments

Four system procedures provide information about segments:

- sp_helpsegment lists the segments in a database or displays information about a particular segment in the database.
- sp_helpdb displays information about the relationship between devices and segments in a database.
- sp_help and sp_helpindex display information about tables and indexes, including the segment to which the object is assigned.

sp_helpsegment

sp_helpsegment, when used without an argument, displays information about all of the segments in the database where you execute it:

```
sp_helpsegment
segment name                          status
------- ----------------------------- ------
      0 system                             0
      1 default                            1
      2 logsegment                         0
      3 seg1                            0
      4 seg2                            0
```

For information about a particular segment, specify the segment name as an argument. Use quotes when requesting information about the default segment:

```
sp_helpsegment "default"
```

The following example displays information about seg1:

```
sp_helpsegment seg1
segment name                           status
------- ------------------------------ ------
      4 seg1                                0

device                 size             free_pages
---------------------- ---------------- -----------
user_data10            15.0MB                  6440
user_data11            15.0MB                  6440
user_data12            15.0MB                  6440

table_name             index_name            indid
---------------------- --------------------- ------
customer               customer                   0

total_size      total_pages free_pages  used_pages
--------------- ----------- ----------- -----------
45.0MB                23040       19320        3720
```

sp_helpdb

When you execute sp_helpdb within a database, and give that database's name, you see information about the segments in the database.

For example:

```
sp_helpdb mydata
```

```
name      db_size    owner  dbid created        status
--------- ---------- ---------- ------------- ------------
mydata        8.0 MB sa        4 May 27, 1993   no options set

device_fragments    size       usage             free kbytes
------------------- ---------- ----------------- -----------
datadev2            4.0 MB     data only                3408
logdev              2.0 MB     log only                 2032
```

```
seg_mydisk1         2.0 MB     data only                2016

device                         segment
------------------------------ -------------------------
datadev2                       default
datadev2                       system
logdev                         logsegment
seg_mydisk1                    seg1
```

sp_help and *sp_helpindex*

When you execute sp_help and sp_helpindex in a database, and give a table's name, you see information about which segment(s) stores the table or its indexes.

For example:

```
    sp_helpindex authors

index_name     index_description                     index_keys
------------   -----------------------------------   ----------
au_index       nonclustered located on seg_mydisk2    au_id
```

Segments and system tables

Three system tables store information about segments: master..sysusages and two system tables in the user database, sysindexes and syssegments. sp_helpsegment uses these tables. Additionally, it finds the database device name in sysdevices.

When you allocate a device to a database with create database or alter database, Adaptive Server adds a row to master..sysusages. The segmap column in sysusages provides bitmaps to the segments in the database for each device.

create database also creates the syssegments table in the user database with these default entries:

```
segment name           status
------- -------------- ------
      0 system             0
      1 default            1
```

```
         2 logsegment         0
```

When you add a segment to a database with sp_addsegment, the procedure:

- Adds a new row to the syssegments table in the user database, and
- Updates the segmap in master..sysusages.

When you create a table or an index, Adaptive Server adds a new row to sysindexes. The segment column in that table stores the segment number, showing where the server will allocate new space for the object. If you do not specify a segment name when you create the object, it is placed on the default segment; otherwise, it is placed on the specified segment.

If you create a table containing text or image columns, a second row is also added to sysindexes for the linked list of text pages; by default, the chain of text pages is stored on the same segment as the table. An example using sp_placeobject to put the text chain on its own segment is included under "A segment tutorial" on page 701.

The name from syssegments is used in create table and create index statements. The status column indicates which segment is the default segment.

Note See "System tables that manage space allocation" on page 664 for more information about the segmap column and the system tables that manage storage.

A segment tutorial

The following tutorial shows how to create a user segment and how to remove all other segment mappings from the device. The examples in this section assume a server using 2K logical page sizes.

When you are working with segments and devices, remember that:

- If you assign space in fragments, each fragment will have an entry in sysusages.
- When you assign an additional fragment of a device to a database, all segments mapped to the existing fragment are mapped to the new fragment.

- If you use alter database to add space on a device that is new to the database, the system and default segments are automatically mapped to the new space.

The tutorial begins with a new database, created with one device for the database objects and another for the transaction log:

```
create database mydata on bigdevice = "4M"
     log on logdev = "2M"
```

Now, if you use mydata, and run sp_helpdb, you see:

```
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created      status
---------- -------- --------- ------ ------------ ---------------
mydata       6.0 MB sa             4 May 27, 1993  no options set

device_fragments       size          usage           free kbytes
---------------------- ------------- --------------- -----------
bigdevice              4.0 MB        data only              3408
logdev                 2.0 MB        log only               2032

device                 segment
---------------------- ----------------------
bigdevice              default
bigdevice              system
logdev                 logsegment
```

Like all newly created databases, mydata has the segments named default, system, and logsegment. Because create database used log on, the logsegment is mapped to its own device, logdev, and the default and system segments are both mapped to bigdevice.

If you add space on the same database devices to mydata, and run sp_helpdb again, you see entries for the added fragments:

```
use master
alter database mydata on bigdevice = "2M"
    log on logdev = "1M"
use mydata
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created      status
---------- -------- --------- ------ ------------ ---------------
mydata       9.0 MB sa             4 May 27, 1993  no options set

device_fragments       size          usage           free kbytes
```

```
---------------------- ------------ -------------- -----------
bigdevice              2.0 MB       data only             2048
bigdevice              4.0 MB       data only             3408
logdev                 1.0 MB       log only              1024
logdev                 2.0 MB       log only              2032

device                 segment
---------------------- ----------------------
bigdevice              default
bigdevice              system
logdev                 logsegment
```

Always add log space to log space and data space to data space. Adaptive Server instructs you to use with override if you try to allocate a segment that is already in use for data to the log, or vice versa. Remember that segments are mapped to entire devices, and not just to the space fragments. If you change any of the segment assignments on a device, you make the change for all of the fragments.

The following example allocates a new database device that has not been used by mydata:

```
use master
alter database mydata on newdevice = 3
use mydata
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created      status
---------- -------- --------- ------ ------------ ---------------
mydata      12.0 MB sa             4 May 27, 1993  no options set

device_fragments       size         usage          free kbytes
---------------------- ------------ -------------- -----------
bigdevice              2.0 MB       data only             2048
bigdevice              4.0 MB       data only             3408
logdev                 1.0 MB       log only              1024
logdev                 2.0 MB       log only              2032
newdevice              3.0 MB       data only             3072

device                 segment
---------------------- ----------------------
bigdevice              default
bigdevice              system
logdev                 logsegment
newdevice              default
newdevice              system
```

The following example creates a segment called new_space on newdevice:

```
sp_addsegment new_space, mydata, newdevice
```

Here is the portion of the sp_helpdb report which lists the segment mapping:

```
device                         segment
---------------------------- ------------------
bigdevice                      default
bigdevice                      system
logdev                         logsegment
newdevice                      default
newdevice                      new_space
newdevice                      system
```

The default and system segments are still mapped to newdevice. If you are planning to use new_space to store a user table or index for improved performance, and you want to ensure that other user objects are not stored on the device by default, reduce the scope of default and system with sp_dropsegment:

```
sp_dropsegment system, mydata, newdevice
sp_dropsegment "default", mydata, newdevice
```

You must include the quotes around “default;” it is a Transact-SQL reserved word.

Here is the portion of the sp_helpdb report that shows the segment mapping:

```
device                         segment
---------------------------- --------------------
bigdevice                      default
bigdevice                      system
logdev                         logsegment
newdevice                      new_space
```

Only new_space is now mapped to newdevice. Users who create objects can use on new_space to place a table or index on the device that corresponds to that segment. Since the default segment is not pointing to that database device, users who create tables and indexes without using the on clause will not be placing them on your specially prepared device.

If you use alter database on newdevice again, the new space fragment acquires the same segment mapping as the existing fragment of that device (that is, the new_space segment only).

At this point, if you use create table and name new_space as the segment, you will get results like these from sp_help and sp_helpsegment:

```
create table mytabl (c1 int, c2 datetime)
    on new_space
sp_help mytabl
```

```
Name                 Owner                Type
-------------------- -------------------- --------------------
mytabl               dbo                  user table

Data_located_on_segment        When_created
------------------------------ --------------------  new_space
May 27 1993  3:21PM

Column_name   Type      Length Nulls Default_name Rule_name
------------- --------- ------ ----- ------------ ----------
c1            int            4     0 NULL           NULL
c2            datetime       8     0 NULL           NULL
Object does not have any indexes.
No defined keys for this object.
```

```
sp_helpsegment new_space

segment name                            status
------- ------------------------------- ------
      3 new_space                            0

device                 size          free_pages
---------------------- ------------- -----------
newdevice              3.0MB                1528

table_name           index_name            indid
-------------------- ---------------------- ------
mytabl                mytabl                     0

total_size      total_pages free_pages  used_pages
--------------- ----------- ----------- -----------
3.0MB                  1536        1528           8
```

Segments and clustered indexes

This example creates a clustered index, without specifying the segment name, using the same table you just created on the new_space segment in the preceding example. Check new_space after create index to verify that no objects remain on the segment:

```
/* Don’t try this at home */
create clustered index mytabl_cix
    on mytabl(c1)
sp_helpsegment new_space
segment name                            status
------- ------------------------------ ------
      3 new_space                           0

device                 size           free_pages
---------------------- -------------- -----------
newdevice              3.0MB                 1528

total_size      total_pages free_pages  used_pages
--------------- ----------- ----------- -----------
3.0MB                  1536        1528           8
```

If you have placed a table on a segment, and you need to create a clustered index, use the on *segment_name* clause, or the table will migrate to the default segment.

CHAPTER 24 Using the *reorg* Command

Update activity against a table can eventually lead to inefficient utilization of space and reduced performance. The reorg command reorganizes the use of table space and improves performance.

This chapter discusses:

reorg subcommands

The reorg command provides four subcommands for carrying out different types and levels of reorganization:

- reorg forwarded_rows undoes row forwarding.
- reorg reclaim_space reclaims unused space left on a page as a result of deletions and row-shortening updates.
- reorg compact both reclaims space and undoes row forwarding.
- reorg rebuild undoes row forwarding and reclaims unused page space, as does reorg compact. In addition, reorg rebuild:
    - Rewrites all rows to accord with a table's clustered index, if it has one

- Writes rows to data pages to accord with any changes made in space management settings through sp_chgattribute
- Drops and re-creates all indexes belonging to the table

The reclaim_space, forwarded_rows, and compact subcommands:

- Minimize interference with other activities by using multiple small transactions of brief duration. Each transaction is limited to eight pages of reorg processing.
- Provide resume and time options that allow you to set a time limit on how long a reorg runs and to resume a reorg from the point at which the previous reorg stopped. This allows you to, for example, use a series of partial reorganizations at off-peak times to reorg a large table. For more information, see “resume and time options for reorganizing large tables” on page 714.

The following considerations apply to the rebuild subcommand:

- reorg rebuild holds an exclusive table lock for its entire duration. On a large table this may be a significant amount of time. However, reorg rebuild accomplishes everything that dropping and re-creating a clustered index does and takes less time. In addition, reorg rebuild rebuilds the table using all of the table’s current space management settings. Dropping and re-creating an index does not use the space management setting for reservepagegap.
- In most cases, reorg rebuild requires additional disk space equal to the size of the table it is rebuilding and its indexes.

The following restrictions hold:

- The table specified in the command, if any, must use either the datarows or datapages locking scheme.
- You must be a System Administrator or the object owner to issue reorg.
- You cannot issue reorg within a transaction.

When to run a *reorg* command

reorg is useful when:

- A large number of forwarded rows causes extra I/O during read operations.
- Inserts and serializable reads are slow because they encounter pages with noncontiguous free space that needs to be reclaimed.
- Large I/O operations are slow because of low cluster ratios for data and index pages.
- sp_chgattribute was used to change a space management setting (reservepagegap, fillfactor, or exp_row_size) and the change is to be applied to all existing rows and pages in a table, not just to future updates.

Using the *optdiag* utility to assess the need for a *reorg*

To assess the need for running a reorg, you can use statistics from the systabstats table and the optdiag utility. systabstats contains statistics on the utilization of table space, while optdiag generates reports based on statistics in both systabstats and the sysstatistics table.

For information on the systabstats table, see the Performance and Tuning Guide. For information about optdiag, see <doctitle>Utility Programs for UNIX Platforms</doctitle>.

Space reclamation without the *reorg* command

Several types of activities reclaim or reorganize the use of space in a table on a page-by-page basis:

- Inserts, when an insert encounters a page that would have enough room if it reclaimed unused space.
- The update statistics command (for index pages only)
- Re-creating clustered indexes
- The housekeeper task, if enable housekeeper GC is set to 1

Each of these has limitations and may be insufficient for use on a large number of pages. For example, inserts may execute more slowly when they need to reclaim space, and may not affect many pages with space that can be reorganized. Space reclamation under the housekeeper task compacts unused space, but it runs only when no other tasks are requesting CPU time, so it may not reach every page that needs it.

Moving forwarded rows to home pages

If an update makes a row too long to fit on its current page, the row is forwarded to another page. A reference to the row is maintained on its original page, the row's *home* page, and all access to the forwarded row goes through this reference. Thus, it always takes two page accesses to get to a forwarded row. If a scan needs to read a large number of forwarded pages, the I/Os caused by extra page accesses slow performance.

reorg forwarded_rows undoes row forwarding by either moving a forwarded row back to its home page, if there is enough space, or by deleting the row and reinserting it in a new home page.

You can get statistics on the number of forwarded rows in a table by querying systabstats and using optdiag.

reorg forwarded_rows syntax

The syntax for reorg forwarded_rows is:

```
reorg forwarded_rows tablename
[with {resume, time = no_of_minutes}]
```

For information about the resume and time options, see "resume and time options for reorganizing large tables" on page 714.

reorg forwarded_rows does not apply to indexes, because indexes do not have forwarded rows.

Using *reorg compact* to remove row forwarding

reorg forwarded_rows uses allocation page hints to find forwarded rows. Because it does not have to search an entire table, this comand executes quickly, but it may miss some forwarded rows. After running reorg forwarded_rows, you can evaluate its effectiveness by using optdiag and checking “Forwarded row count.” If “Forwarded row count” is high, you can then run reorg compact, which goes through a table page by page and undoes all row forwarding.

Reclaiming unused space from deletes and updates

When a task performs a delete operation or an update that shortens row length, the empty space is preserved in case the transaction is rolled back. If a table is subject to frequent deletes and row-shortening updates, unreclaimed space may accumulate to the point that it impairs performance.

reorg reclaim_space reclaims unused space left by deletes and updates. On each page that has space resulting from committed deletes or row-shortening updates, reorg reclaim_space rewrites the remaining rows contiguously, leaving all the unused space at the end of the page. If all rows have been deleted and there are no remaining rows, reorg reclaim_space deallocates the page.

You can get statistics on the number of unreclaimed row deletions in a table from the systabstats table and by using the optdiag utility. There is no direct measure of how much unused space there is as a result of row-shortening updates.

reorg reclaim_space syntax

The syntax for reorg reclaim_space is:

```
reorg reclaim_space tablename [indexname]
    [with {resume, time = no_of_minutes}]
```

If you specify only a table name, only the table’s data pages are reorganized to reclaim unused space; in other words, indexes are not affected. If you specify an index name, only the pages of the index are reorganized.

For information about the resume and time options, see “resume and time options for reorganizing large tables” on page 714.

Reclaiming unused space and undoing row forwarding

reorg compact combines the functions of reorg reclaim_space and reorg forwarded_rows. Use reorg compact when:

- You don't need to rebuild an entire table (reorg rebuild); however, both row forwarding and unused space from deletes and updates may be affecting performance.
- There are a large number of forwarded rows. See "Using reorg compact to remove row forwarding" on page 711.

reorg compact syntax

The syntax for reorg compact is:

```
reorg compact tablename
    [with {resume, time = no_of_minutes}]
```

For information about the resume and time options, see "resume and time options for reorganizing large tables" on page 714.

Rebuilding a table

Use reorg rebuild when:

- Large I/O is not being selected for queries where it is usually used, and optdiag shows a low cluster ratio for datapages, data rows, or index pages.
- You used sp_chgattribute to change one or more of the exp_row_size, reservepagegap, or fillfactor space management settings and you want the changes to apply not only to future data, but also to existing rows and pages. For information about sp_chgattribute, see the Adaptive Server Reference Manual.

If a table needs to be rebuilt because of a low cluster ratio, it may also need to have its space management settings changed (see "Changing space management settings before using reorg rebuild" on page 713).

reorg rebuild uses a table's current space management settings to rewrite the rows in the table according to the table's clustered index, if it has one. All indexes on the table are dropped and re-created using the current space management values for reservepagegap and fillfactor. After a rebuild, a table has no forwarded rows and no unused space from deletions or updates.

reorg rebuild syntax

The syntax for reorg rebuild is:

reorg rebuild *tablename*

Prerequisites for running *reorg rebuild*

Before you run reorg rebuild on a table:

- Set the database option select into/bulkcopy/pllsort to true and run checkpoint in the database.
- Make sure that additional disk space, equal to the size of the table and its indexes, is available.

To set select into/bulkcopy/pllsort to true and checkpoint the database, use the following isql commands:

```
1> use master
2> go
1> sp_dboption pubs2,
    "select into/bulkcopy/pllsort", true
2> go
1> use pubs2
2> go
1> checkpoint
2> go
```

Following a rebuild on a table:

- You must dump the database containing the table before you can dump the transaction log.
- Distribution statistics for the table are updated.
- All stored procedures that reference the table will be recompiled the next time they are run.

Changing space management settings before using *reorg rebuild*

When reorg rebuild rebuilds a table, it rewrites all table and index rows according to the table's current settings for reservepagegap, fillfactor, and exp_row_size. These properties all affect how quickly inserts cause a table to become fragmented, as measured by a low cluster ratio.

If it appears that a table quickly becomes fragmented and needs to be rebuilt too frequently, it may be a sign that you need to change the table's space management settings before you run reorg rebuild.

To change the space management settings, use sp_chgattribute (see the Adaptive Server Reference Manual). For information on space management settings, see Performance and Tuning Guide

resume and *time* options for reorganizing large tables

Use the resume and time options of the reorg command when reorganizing an entire table would take too long and interfere with other database activities. time allows you to run a reorg for a specified length of time. resume allows you to start a reorg at the point in a table where the previous reorg left off. In combination, the two options allow you to reorganize a large table by running a series of partial reorganizations (for example, during off-hours).

resume and time not available with reorg rebuild.

Syntax for using *resume* and *time* in *reorg* commands

The syntax for resume and time is:

```
reorg reclaim_space tablename [indexname]
    [with {resume, time = no_of_minutes}]

reorg forwarded_rows tablename
    [with {resume, time = no_of_minutes}]

reorg compact tablename
    [with {resume, time = no_of_minutes}]
```

The following considerations apply:

- If you specify only the resume option, the reorg begins at the point where the previous reorg stopped and continues to the end of the table.
- If you specify only the time option, the reorg starts at the beginning of the table and continues for the specified number of minutes.
- If you specify both options, the reorg starts at the point where the previous reorg stopped and continues for the specified number of minutes.

Specifying *no_of_minutes* in the *time* option

The *no_of_minutes* argument in the time option refers to elapsed time, not CPU time. For example, to run reorg compact for 30 minutes, beginning where a previous reorg compact finished, enter:

```
reorg compact tablename with resume, time=30
```

If the reorg process goes to sleep during any part of the 30 minutes, it still counts as part of the elapsed time and does not add to the duration of the reorg.

When the amount of time specified has passed, reorg saves statistics about the portion of the table or index that was processed in the systabstats table. This information is used as the restart point for a reorg with the resume option. The restart points for each of the three subcommands that take resume and time options are maintained separately. You cannot, for example, start a reorg with reorg reclaim_space and then resume it with reorg compact.

If you specify *no_of_minutes*, and reorg arrives at the end of a table or an index before the time is up, it returns to the beginning of the object and continues until it reaches its time limit.

Note resume and time allow you to reorganize an entire table or index over multiple runs. However, if there are updates between reorg runs, some pages may be processed twice and some pages may not be processed at all.

Using the *reorg rebuild* command on indexes

The reorg rebuild command allows you to rebuild individual indexes while the table itself is accessible for read and update activities.

Syntax

The syntax for rebuilding an index is:

```
reorg rebuild indexname
```

Comments

To use reorg rebuild, you must be the table owner or the Database Owner, or have System Administrator privileges.

If you omit the index name, the entire table is rebuilt.

If you specify an index, only that index is rebuilt.

Requirements for using reorg rebuild on an index are less stringent than for tables. The following rules apply:

- You do not need to set select into to rebuild an index.
- Rebuilding a table requires space for a complete copy of the table. Rebuilding an index works in small transactions, and deallocates pages once they are copied; therefore, the process only needs space for the pages copied on each transaction.
- You can rebuild the index on a table while transaction level scans (dirty reads) are active.

Limitations

The reorg command applies only to tables using datarows or datapages locking. You cannot run reorg on a table that uses allpages locking.

You cannot run reorg on a text index, the name from sysindexes associated with a text chain.

You cannot run reorg within a transaction.

You can do a dump tran on a table after rebuilding its index. However, you cannot do a dump tran if the entire table has been rebuilt.

You can rebuild the index for systabstats, but you cannot run reorg rebuild on the table itself.

Although online index rebuilding is allowed on a placement index, it rebuilds only the index pages. The data pages remain untouched, which means datarows are neither sorted nor rewritten to fresh pages. You can rebuild data pages by dropping a placement index, and then re-creating it.

How indexes are rebuilt with *reorg rebuild indexname*

Rebuilding a single index rewrites all index rows to new pages. This improves performance by:

- Improving clustering of the leaf level of the index
- Applying stored values for the fill factor on the index, which can reduce page splits

- Applying any stored value for *reservepagegap*, which can help reserve pages for future splits

To reduce contention with users whose queries need to use the index, reorg rebuild locks a small number of pages at a time. Rebuilding an index is a series of independent transactions, with some independent, nested transactions. Approximately 32 pages are rebuilt in each nested transaction and approximately 256 pages are rebuilt in each outer transaction. Address locks are acquired on the pages being modified and are released at the end of the topaction. The pages deallocated in a transaction are not available for reuse until the next transaction begins.

If the reorg rebuild command stops running, the transactions that are already committed are not rolled back. Therefore, the part that has been reorganized is well clustered with desired space utilization, and the part that has not been reorganized is the same as it was before you ran the command. The index remains logically consistent.

Note Rebuilding the clustered index does not affect the data pages of the table. It only affects the leaf pages and higher index levels. Non-leaf pages above level 1 are not rebuilt.

Space requirements for rebuilding an index

If you do not specify fill_factor or reservepagegap, rebuilding an index requires additional space of aproximately 256 pages or less in the data segment. The amount of log space required is larger than that required to drop the index and re-create it using create index, but it should be only a small fraction of the actual index size. The more additional free space is available, the better the index clustering will be.

Note reorg rebuild may not rebuild those parts of the index that are already well clustered and have the desired space utilization.

Performance characteristics

Index scans are faster after you run reorg.

Running reorg against a table can have a negative effect on performance of concurrent queries.

Status messages

Running reorg rebuild *indexname* on a large table may take a long time. Periodic status messages are printed to give the user an idea of how reorg has progressed. Starting and ending messages are written to the error log and to the client process executing reorg. In-progress messages go only to the client.

A status reporting interval is calculated as either 10% of the pages to be processed or 10,000 pages, whichever is larger. When this number of pages is processed, a status message is printed. Therefore, no more than 10 messages are printed, regardless of the size of the index. Status messages for existing reorg commands are printed more frequently.

CHAPTER 25 Checking Database Consistency

This chapter describes how to check database consistency and perform some kinds of database maintenance using the dbcc commands.

Topics covered in this chapter include:

What is the database consistency checker?

The database consistency checker (dbcc) provides commands for checking the logical and physical consistency of a database. Two major functions of dbcc are:

- Checking page linkage and data pointers at both the page level and the row level using checkstorage or checktable and checkdb
- Checking page allocation using checkstorage, checkalloc, checkverify, tablealloc, and indexalloc

dbcc checkstorage stores the results of checks in the dbccdb database. You can print reports from dbccdb using the dbcc stored procedures.

Use the dbcc commands:

- As part of regular database maintenance – the integrity of the internal structures of a database depends upon the System Administrator or Database Owner running database consistency checks on a regular basis.
- To determine the extent of possible damage after a system error has occurred.
- Before backing up a database for additional confidence in the integrity of the backup.
- If you suspect that a database is damaged – for example, if using a particular table generates the message "Table corrupt," you can use dbcc to determine if other tables in the database are also damaged.

If you are using Component Integration Services, there are additional dbcc commands you can use for remote databases. For more information, see the *Component Integration Services User's Guide*.

Understanding page and object allocation concepts

When you initialize a database device, the disk init command divides the new space into **allocation units**. The size of the allocation unit depends on the size of the logical pages your server uses (2, 4, 8, or 16K). The first page of each allocation unit is an **allocation page**, which tracks the use of all pages in the allocation unit. Allocation pages have an object ID of 99; they are not real database objects and do not appear in system tables, but dbcc errors on allocation pages report this value.

When a table or an index requires space, Adaptive Server allocates a block of 8 pages to the object. This 8-page block is called an *extent*. Each allocation unit contains 32 extents. The size of the extent also depends on the size of the server logical pages. Adaptive Server uses extents as a unit of space management to allocate and deallocate space as follows:

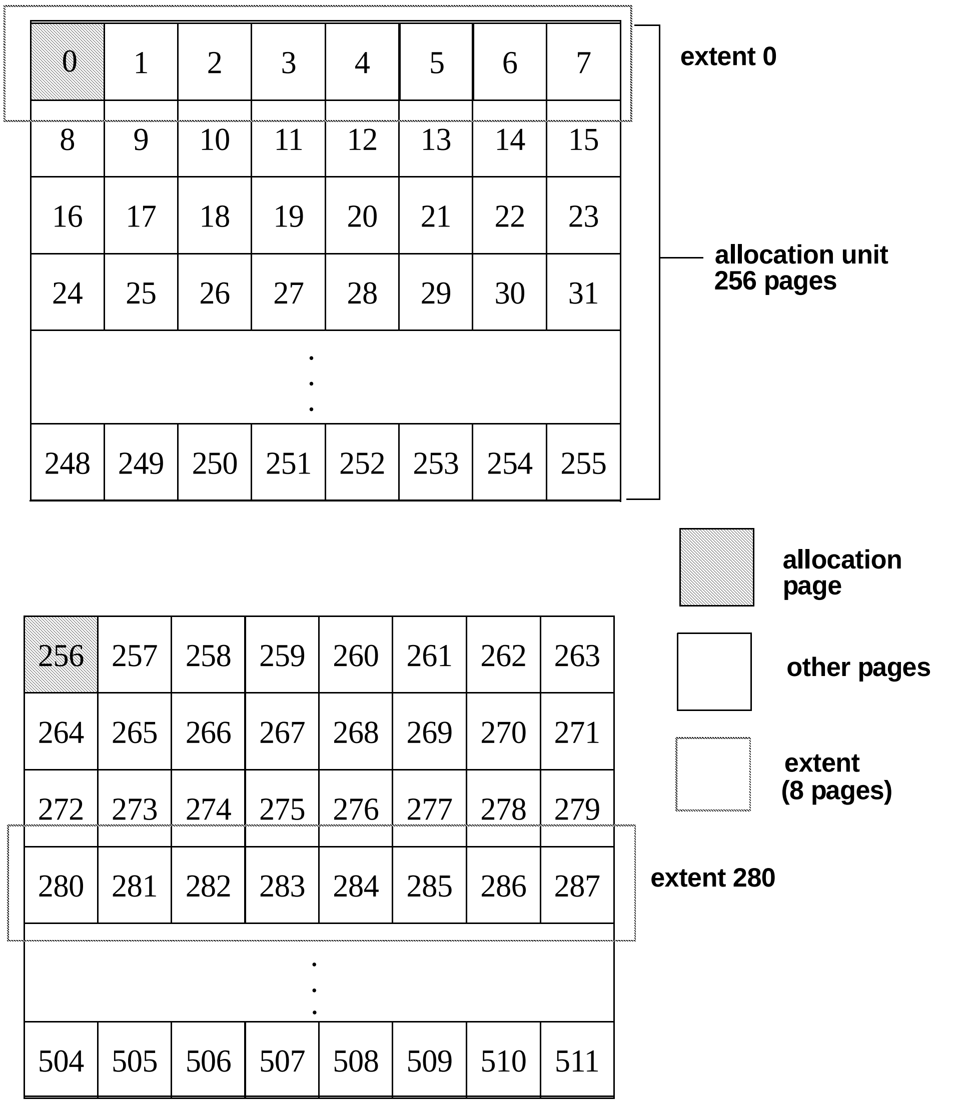
- When you create a table or an index, Adaptive Server allocates an extent for the object.

- When you add rows to an existing table, and the existing pages are full, Adaptive Server allocates another page. If all pages in an extent are full, Adaptive Server allocates an additional extent.
- When you drop a table or an index, Adaptive Server deallocates the extents it occupied.
- When you delete rows from a table so that it shrinks by a page, Adaptive Server deallocates the page. If the table shrinks off the extent, Adaptive Server deallocates the extent.

Every time space is allocated or deallocated on an extent, Adaptive Server records the event on the allocation page that tracks the extents for that object. This provides a fast method for tracking space allocations in the database, since objects can shrink or grow without excess overhead.

Figure 25-1 shows how data pages are set up within extents and allocation units in Adaptive Server databases.

Figure 25-1: Page management with extents



dbcc checkalloc checks all allocation pages (page 0 and all pages divisible by 256) in a database and reports on the information it finds. dbcc indexalloc and dbcc tablealloc check allocation for specific database objects.

Understanding the object allocation map (OAM)

Each table and index on a table has an *Object Allocation Map* (*OAM*). The OAM is stored on pages allocated to the table or index and is checked when a new page is needed for the index or table. A single OAM page can hold allocation mapping for between 2,000 and 63,750 data or index pages.

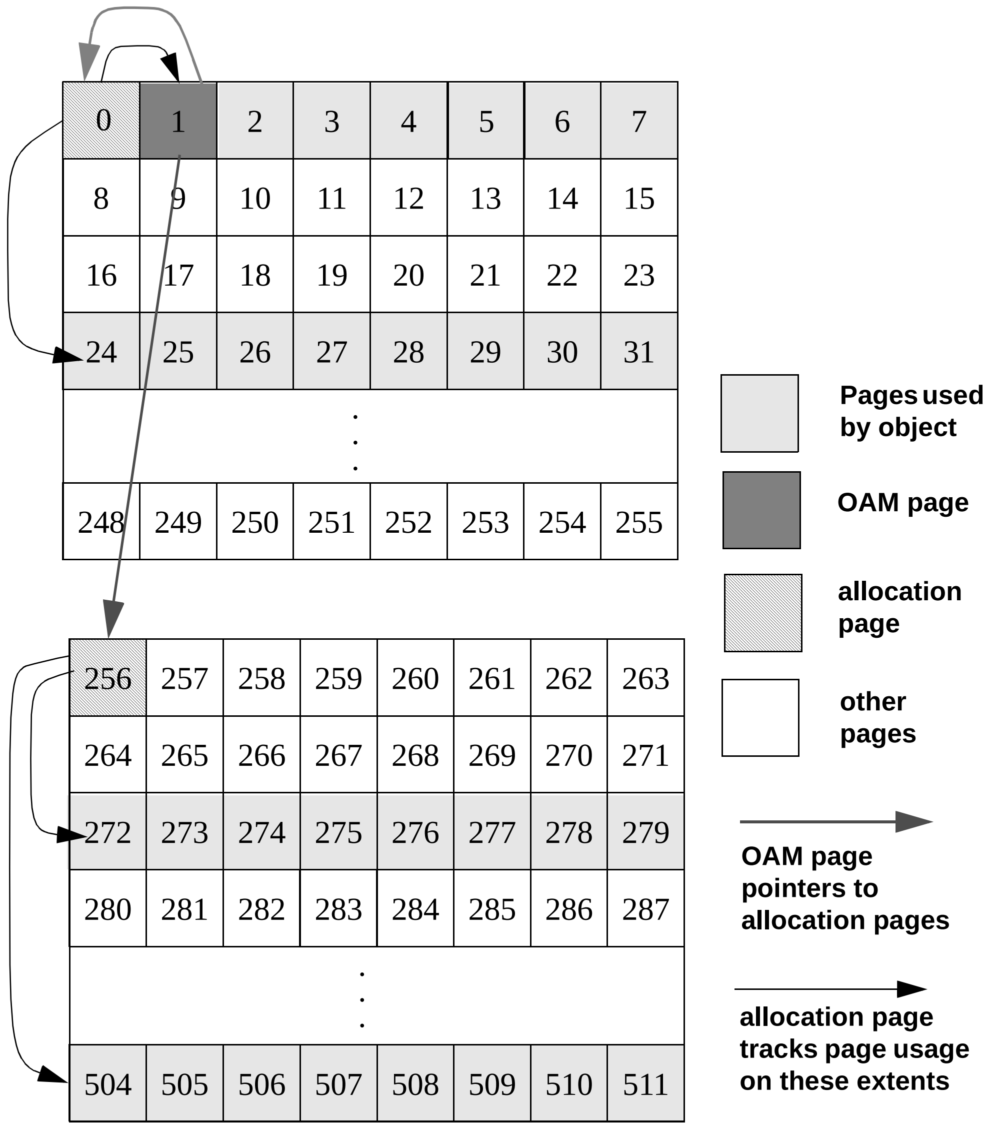
The OAM pages point to the allocation page for each allocation unit where the object uses space. The allocation pages, in turn, track the information about extent and page usage within the allocation unit. In other words, if the titles table is stored on extents 24 and 272, the OAM page for the titles table points to pages 0 and 256.

Figure 25-2 shows an object stored on 4 extents, numbered 0, 24, 272 and 504 for a server that uses 2K logical pages. The OAM is stored on the first page of the first segment. In this case, since the allocation page occupies page 0, the OAM is located on page 1.

This OAM points to two allocation pages: page 0 and page 256.

These allocation pages track the pages used in each extent used by all objects with storage space in the allocation unit. For the object in this example, it tracks the allocation and de-allocation of pages on extents 0, 24, 272, and 504.

Figure 25-2: OAM page and allocation page pointers

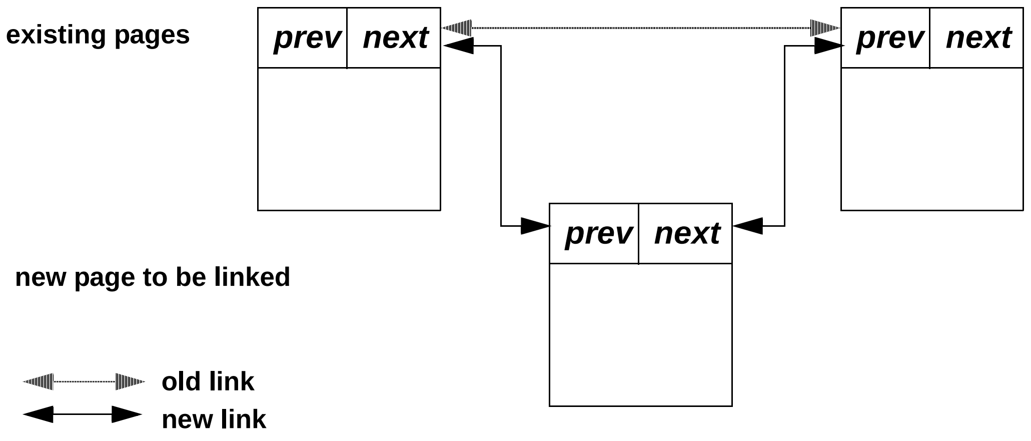


dbcc checkalloc and dbcc tablealloc examine this OAM page information, in addition to checking page linkage, as described in "Understanding page linkage" on page 725.

Understanding page linkage

After a page has been allocated to a table or an index, that page is linked with other pages used for the same object. Figure 25-3 illustrates this linking. Each page contains a header that includes the number of the page that precedes it ("prev") and of the page that follows it ("next"). When a new page is allocated, the header information on the surrounding pages changes to point to that page. dbcc checktable and dbcc checkdb check page linkage. dbcc checkalloc, tablealloc, and indexalloc compare page linkage to information on the allocation page.

Figure 25-3: How a newly allocated page is linked with other pages



What checks can be performed with *dbcc*?

Table 25-1 summarizes the checks performed by the dbcc commands. Table 25-2 on page 738 compares the different dbcc commands.

Table 25-1: Comparison of checks performed by dbcc commands

Checks performed	*check-storage*	*check-table*	*check-db*	*check-alloc*	*index-alloc*	*table-alloc*	*check-catalog*
Checks allocation of text valued columns	X						
Checks index consistency		X	X				
Checks index sort order		X	X				

Checks performed	***check-storage***	***check-table***	***check-db***	***check-alloc***	***index-alloc***	***table-alloc***	***check-catalog***
Checks OAM page entries	X	X	X		X	X	
Checks page allocation	X			X	X	X	
Checks page consistency	X	X	X				
Checks pointer consistency	X	X	X				
Checks system tables							X
Checks text column chains	X	X	X	X			
Checks text valued columns	X	X	X				

Note You can run all dbcc commands except dbrepair and checkdb with the fix option while the database is active.

Only the table owner can execute dbcc with the checktable, fix_text, or reindex keywords. Only the Database Owner can use the checkstorage, checkdb, checkcatalog, checkalloc, indexalloc, and tablealloc keywords. Only a System Administrator can use the dbrepair keyword.

Checking consistency of databases and tables

The dbcc commands for checking the consistency of databases and tables are:

- dbcc checkstorage
- dbcc checktable
- dbcc checkdb

dbcc checkstorage

Use dbcc checkstorage to perform the following checks:

- Allocation of text valued columns
- Page allocation and consistency
- OAM page entries
- Pointer consistency

- Text valued columns and text column chains

The syntax for dbcc checkstorage is:

```
dbcc checkstorage [(dbname)]
```

where *dbname* is the name of the **target database** (the database to be checked).

Advantages of using *dbcc checkstorage*

The dbcc checkstorage command:

- Combines many of the checks provided by the other dbcc commands
- Does not lock tables or pages for extended periods, which allows dbcc to locate errors accurately while allowing concurrent update activity
- Scales linearly with the aggregate I/O throughput
- Separates the functions of checking and reporting, which allows custom evaluation and report generation
- Provides a detailed description of space usage in the target database
- Records dbcc checkstorage activity and results in the dbccdb database, which allows trend analysis and provides a source of accurate diagnostic information

Comparison of *dbcc checkstorage* and other *dbcc* commands

dbcc checkstorage is different from the other dbcc commands in that it requires:

- The dbccdb database to store configuration information and the results of checks made on the target database. For more information, see “Preparing to use dbcc checkstorage” on page 748.
- At least two workspaces to use during the check operation. See “dbccdb Workspaces” on page 87 in the *Adaptive Server Reference Manual*.
- System and stored procedures to help you prepare your system to use dbcc checkstorage and to generate reports on the data stored in dbccdb. See “Preparing to use dbcc checkstorage” on page 748, “Maintaining dbccdb” on page 759, and “Generating reports from dbccdb” on page 762.

dbcc checkstorage does not repair any faults. After you run dbcc checkstorage and generate a report to see the faults, you can run the appropriate dbcc command to repair the faults.

Understanding the *dbcc checkstorage* operation

The dbcc checkstorage operation consists of the following steps:

1. Inspection – dbcc checkstorage uses the device allocation and the segment definition of the database being checked to determine the level of parallel processing that can be used. dbcc checkstorage also uses the configuration parameters max worker processes and dbcc named cache to limit the level of parallel processing that can be used.
2. Planning – dbcc checkstorage generates a plan for executing the operation that takes advantage of the parallelism discovered in step 1.
3. Execution and optimization – dbcc checkstorage uses Adaptive Server worker processes to perform parallel checking and storage analysis of the target database. It attempts to equalize the work performed by each worker process and consolidates the work of underutilized worker processes. As the check operation proceeds, dbcc checkstorage extends and adjusts the plan generated in step 2 to take advantage of the additional information gathered during the check operation.
4. Reporting and control – during the check operation, dbcc checkstorage records in the dbccdb database all the faults it finds in the target database for later reporting and evaluation. It also records the results of its storage analysis in dbccdb. When dbcc checkstorage encounters a fault, it attempts to recover and continue the operation, but ends operations that cannot succeed after the fault. For example, a defective disk does not cause dbcc checkstorage to fail; however, check operations performed on the defective disk cannot succeed, so they are not performed.

   If another session performs drop table concurrently, dbcc checkstorage might fail in the initialization phase. If this happens, run dbcc checkstorage again when the drop table process is finished.

Performance and scalability

dbcc checkstorage scales linearly with aggregate I/O throughput for a substantial performance improvement over dbcc checkalloc. The scaling property of dbcc checkstorage means that if the database doubles in size and the hardware doubles in capacity (realizable I/O throughput), the time required for a dbcc check remains unchanged. Doubling the capacity would typically mean doubling the number of disk spindles and providing sufficient additional I/O channel capacity, system bus capacity, and CPU capacity to realize the additional aggregate disk throughput.

Most of the checks performed by using dbcc checkalloc and dbcc checkdb, including text column chain verification, are achieved with a single check when you use dbcc checkstorage, thereby eliminating redundant check operations.

dbcc checkstorage checks the entire database, including unused pages, so execution time is relative to database size. Therefore, when you use dbcc checkstorage, there is not a large difference between checking a database that is nearly empty and checking one that is nearly full, as there is with the other dbcc commands.

Unlike the other dbcc commands, the performance of dbcc checkstorage does not depend heavily on data placement. Therefore, performance is consistent for each session, even if the data placement changes between sessions.

Because dbcc checkstorage does extra work to set up the parallel operation and records large amounts of data in dbccdb, the other dbcc commands are faster when the target database is small.

The location and allocation of the workspaces used by dbcc checkstorage can affect performance and scalability. For more information on how to set up the workspaces to maximize the performance and scalability of your system, see “dbccdb Workspaces” on page 87 in the *Adaptive Server Reference Manual*.

To run dbcc checkstorage and one of the system procedures for generating reports with a single command, use sp_dbcc_runcheck. For information on the report generating system procedures, see “Generating reports from dbccdb” on page 762.

dbcc checktable

dbcc checktable checks the specified table to see that:

- Index and data pages are linked correctly
- Indexes are sorted properly
- Pointers are consistent
- Data rows on each page have entries in the row-offset table; these entries match the locations for the data rows on the page
- Data rows on each page have entries in the row-offset table in the page that match their respective locations on the page
- Partition statistics for partitioned tables are correct

The syntax for dbcc checktable is:

```
dbcc checktable ({table_name | table_id}
    [, skip_ncindex] )
```

The skip_ncindex option allows you to skip checking the page linkage, pointers, and sort order on nonclustered indexes. The linkage and pointers of clustered indexes and data pages are essential to the integrity of your tables. You can drop and re-create nonclustered indexes if Adaptive Server reports problems with page linkage or pointers.

When checkstorage returns a fault code of 100035, and checkverify confirms that the spacebit fault is a hard fault, you can use dbcc checktable to fix the reported fault.

The syntax is:

```
dbcc checktable (table_name, fix_spacebits)
```

where *table_name* is the name of the table to repair.

dbcc checktable can be used with the table name or the table's object ID. The sysobjects table stores this information in the name and id columns.

The following example shows a report on an undamaged table:

```
dbcc checktable(titles)
go
Checking titles
The total number of data pages in this table is 3.
Table has 18 data rows.
DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator
(SA) role.
```

If the table is partitioned, dbcc checktable checks data page linkage and partition statistics for each partition. For example:

```
dbcc checktable(historytab)
go
Checking historytab
The total number of pages in partition 1 is 20.
The total number of pages in partition 2 is 17.
The total number of pages in partition 3 is 19.
The total number of pages in partition 4 is 17.
The total number of pages in partition 5 is 20.
The total number of pages in partition 6 is 16.
The total number of pages in partition 7 is 19.
The total number of pages in partition 8 is 17.
The total number of pages in partition 9 is 19.
The total number of pages in partition 10 is 16.
The total number of data pages in this table is 190.
Table has 1536 data rows.
DBCC execution completed. If DBCC printed error
messages, contact a user with System Administrator
(SA) role.
```

To check a table that is not in the current database, supply the database name. To check a table owned by another object, supply the owner's name. You must enclose any qualified table name in quotes. For example:

```
dbcc checktable("pubs2.newuser.testtable")
```

dbcc checktable addresses the following problems:

- If the page linkage is incorrect, dbcc checktable displays an error message.
- If the sort order (sysindexes.soid) or character set (sysindexes.csid) for a table with columns with char or varchar datatypes is incorrect, and the table's sort order is compatible with Adaptive Server's default sort order, dbcc checktable corrects the values for the table. Only the binary sort order is compatible across character sets.

Note If you change sort orders, character-based user indexes are marked "read-only" and must be checked and rebuilt, if necessary. See Chapter 7, "Configuring Character Sets, Sort Orders, and Languages," for more information about changing sort orders.

- If data rows are not accounted for in the first OAM page for the object, dbcc checktable updates the number of rows on that page. This is not a serious problem. The built-in function rowcnt uses this value to provide fast row estimates in procedures like sp_spaceused.

You can improve dbcc checktable performance by using enhanced page fetching.

dbcc checkdb

dbcc checkdb runs the same checks as dbcc checktable on each table in the specified database. If you do not give a database name, dbcc checkdb checks the current database. dbcc checkdb gives similar messages to those returned by dbcc checktable and makes the same types of corrections.

The syntax for dbcc checkdb is:

```
dbcc checkdb [(database_name [, skip_ncindex]) ]
```

If you specify the optional skip_ncindex, dbcc checkdb does not check any of the nonclustered indexes on user tables in the database.

Checking page allocation

The dbcc commands that you use to check page allocation are:

- dbcc checkalloc
- dbcc indexalloc
- dbcc tablealloc

dbcc checkalloc

dbcc checkalloc ensures that:

- All pages are correctly allocated.
- Partition statistics on the allocation pages are correct.
- No page is allocated that is not used.
- No page is used that is not allocated.

The syntax for dbcc checkalloc is:

```
dbcc checkalloc [(database_name [, fix | nofix] )]
```

If you do not provide a database name, dbcc checkalloc checks the current database.

With the fix option, dbcc checkalloc can fix all allocation errors that would otherwise be fixed by dbcc tablealloc and can also fix pages that remain allocated to objects that have been dropped from the database. Before you can use dbcc checkalloc with the fix option, you must put the database into single-user mode. For details on using the fix and no fix options, see “Correcting allocation errors using the fix | nofix option” on page 735.

dbcc checkalloc output consists of a block of data for each table, including the system tables and the indexes on each table. For each table or index, it reports the number of pages and extents used. Table information is reported as either INDID=0 or INDID=1. Tables without clustered indexes have INDID=0, as shown in the example report on the salesdetail table. Tables with clustered indexes have INDID=1. The report for these indexes includes information at both the data and index level, as shown in the example reports on titleauthor and titles. Nonclustered indexes are numbered consecutively, starting with INDID=2.

The following report on pubs2 shows the output for the salesdetail, titleauthor, and titles tables:

```
***************************************************************
TABLE: salesdetail              OBJID = 144003544
INDID=0  FIRST=297      ROOT=299        SORT=0
        Data level: 0.  3 Data  Pages in 1 extents.
INDID=2  FIRST=289      ROOT=290        SORT=1
        Indid     : 2.  3 Index Pages in 1 extents.
INDID=3  FIRST=465      ROOT=466        SORT=1
        Indid     : 3.  3 Index Pages in 1 extents.
TOTAL # of extents = 3
***************************************************************
TABLE: titleauthor              OBJID = 176003658
INDID=1  FIRST=433      ROOT=425        SORT=1
        Data level: 1.  1 Data  Pages in 1 extents.
        Indid     : 1.  1 Index Pages in 1 extents.
INDID=2  FIRST=305      ROOT=305        SORT=1
        Indid     : 2.  1 Index Pages in 1 extents.
INDID=3  FIRST=441      ROOT=441        SORT=1
        Indid     : 3.  1 Index Pages in 1 extents.
TOTAL # of extents = 4
***************************************************************
TABLE: titles           OBJID = 208003772
INDID=1  FIRST=417      ROOT=409        SORT=1
        Data level: 1.  3 Data  Pages in 1 extents.
        Indid     : 1.  1 Index Pages in 1 extents.
INDID=2  FIRST=313       ROOT=313        SORT=1
        Indid     : 2.  1 Index Pages in 1 extents.
TOTAL # of extents = 3
***************************************************************
```

dbcc indexalloc

dbcc indexalloc checks the specified index to see that:

- All pages are correctly allocated.
- No page is allocated that is not used.
- No page is used that is not allocated.

dbcc indexalloc is an index-level version of dbcc checkalloc, providing the same integrity checks on an individual index. You can specify either the table name or the table's object ID (the id column in sysobjects) and the index's indid in sysindexes. dbcc checkalloc and dbcc indexalloc include the index IDs in their output.

The syntax for dbcc indexalloc is:

```
dbcc indexalloc ( {table_name | table_id }, index_id
    [, {full | optimized | fast | null}
    [, fix | nofix]])
```

If you want to use the fix or nofix option for dbcc indexalloc, you must also specify one of the report options (full, optimized, fast, or null). For details on using the fix and no fix options, see "Correcting allocation errors using the fix | nofix option" on page 735. For details on the reports, see "Generating reports with dbcc tablealloc and dbcc indexalloc" on page 736.

You can run sp_indsuspect to check the consistency of sort order in indexes and dbcc reindex to repair inconsistencies. For details see "Using sp_indsuspect to find corrupt indexes" on page 283 and "Rebuilding indexes after changing the sort order" on page 283.

dbcc tablealloc

dbcc tablealloc checks the specified user table to ensure that:

- All pages are correctly allocated.

- Partition statistics on the allocation pages are correct.
- No page is allocated that is not used.
- No page is used that is not allocated.

The syntax for dbcc tablealloc is:

```
dbcc tablealloc ({table_name | table_id}
    [, {full | optimized | fast | null}
    [, fix | nofix]])
```

You can specify either the table name or the table's object ID from the id column in sysobjects.

If you want to use the fix or nofix options for dbcc tablealloc, you must also specify one of the report options (full, optimized, fast, or null). For details on using the fix and no fix options, see "Correcting allocation errors using the fix | nofix option" on page 735. For details on the reports, see "Generating reports with dbcc tablealloc and dbcc indexalloc" on page 736.

Correcting allocation errors using the *fix* | *nofix* option

You can use the fix | nofix option with dbcc checkalloc, dbcc tablealloc, and dbcc indexalloc. It specifies whether or not the command fixes the allocation errors in tables. The default for all user tables is fix. The default for all system tables is nofix.

Before you can use the fix option on system tables, you must put the database into single-user mode:

```
sp_dboption dbname, "single user", true
```

You can issue this command only when no one is using the database. While it is in effect, only the user who issued it can access the database. Because of this, we recommend that you run dbcc checkalloc with nofix, so that the database is available to other users, and then use dbcc tablealloc or dbcc indexalloc with fix to correct errors in individual tables or indexes.

Output from dbcc tablealloc with fix displays allocation errors and any corrections that were made. The following example shows an error message that appears whether or not the fix option is used:

```
Msg 7939, Level 22, State 1:
Line 2:
```

```
Table Corrupt: The entry is missing from the OAM for
object id 144003544 indid 0 for allocation page 2560.
```

When you use fix, the following message indicates that the missing entry has been restored:

```
The missing OAM entry has been inserted.
```

The fix|nofix option works the same in dbcc indexalloc as it does in dbcc tablealloc.

Generating reports with *dbcc tablealloc* and *dbcc indexalloc*

You can generate three types of reports with dbcc tablealloc or dbcc indexalloc:

- full – produces a report containing all types of allocation errors. Using the full option with dbcc tablealloc gives the same results as using dbcc checkalloc at a table level.
- optimized – produces a report based on the allocation pages listed in the OAM pages for the table. When you use the optimized option, dbcc tablealloc does not report and cannot fix unreferenced extents on allocation pages that are not listed in the OAM pages. If you do not specify a report type, or if you specify null, optimized is the default.
- fast – produces an exception report of pages that are referenced but not allocated in the extent (2521-level errors); does not produce an allocation report.

For a comparison of speed, completeness, locking, and performance issues for these options and other dbcc commands, see Table 25-2 on page 738.

Checking consistency of system tables

dbcc checkcatalog checks for consistency within and between the system tables in a database. For example, it verifies that:

- Every type in syscolumns has a matching entry in systypes.

- Every table and view in sysobjects has at least one column in syscolumns.
- The last checkpoint in syslogs is valid.

It also lists the segments defined for use by the database.

The syntax for dbcc checkcatalog is:

```
dbcc checkcatalog [(database_name)]
```

If you do not specify a database name, dbcc checkcatalog checks the current database.

```
dbcc checkcatalog (testdb)
```

```
Checking testdb
The following segments have been defined for database 5 (database name testdb).
virtual start addr      size      segments
--------------------    ------    --------------------------
33554432                4096       0
                                   1
16777216                102        2
DBCC execution completed. If DBCC printed error messages, see your System
Administrator.
```

Strategies for using consistency checking commands

The following sections compare the performance of the dbcc commands, provide suggestions for scheduling and strategies to avoid serious performance impacts, and provide information about dbcc output.

Comparing the performance of *dbcc* commands

Table 25-2 compares the speed, thoroughness, the level of checking and locking, and performance implications of the dbcc commands. Remember that dbcc checkdb, dbcc checktable, and dbcc checkcatalog perform different types of integrity checks than dbcc checkalloc, dbcc tablealloc, and dbcc indexalloc. dbcc checkstorage performs a combination of the some of the checks performed by the other commands. Table 25-1 on page 725 shows which checks are performed by the commands.

Table 25-2: Comparison of the performance of dbcc commands

Command and option	Level	Locking and performance	Speed	Thorough-ness
checkstorage	Page chains and data rows for all indexes, allocation pages, OAM pages, device and partition statistics	No locking; performs extensive I/O and may saturate the system's I/O; can use dedicated cache with minimal impact on other caches	Fast	High
checktable checkdb	Page chains, sort order, data rows, and partition statistics for all indexes	Shared table lock; dbcc checkdb locks one table at a time and releases the lock after it finishes checking that table	Slow	High
checktable checkdbwith skip_ncindex	Page chains, sort order, and data rows for tables and clustered indexes	Shared table lock; dbcc checkdb locks one table at a time and releases the lock after it finishes checking that table	Up to 40% faster than without skip_ncindex	Medium
checkalloc	Page chains and partition statistics	No locking; performs extensive I/O and may saturate the I/O calls; only allocation pages are cached	Slow	High
tablealloc full indexalloc fullwith full	Page chains	Shared table lock; performs extensive I/O; only allocation pages are cached	Slow	High
tablealloc indexallocwith optimized	Allocation pages	Shared table lock; performs extensive I/O; only allocation pages are cached	Moderate	Medium
tablealloc indexallocwith fast	OAM pages	Shared table lock	Fast	Low
checkcatalog	Rows in system tables	Shared page locks on system catalogs; releases lock after each page is checked; very few pages cached	Moderate	Medium

Using large I/O and asynchronous prefetch

Some dbcc commands can use large I/O and asynchronous prefetch when these are configured for the caches used by the databases or objects to be checked.

dbcc checkdb and dbcc checktable use large I/O pools for the page chain checks on tables when the tables use a cache with large I/O configured. The largest I/O size available is used. When checking indexes, dbcc uses only 2K buffers.

The dbcc checkdb, dbcc checktable, and the dbcc allocation checking commands, checkalloc, tablealloc and indexalloc, use asynchronous prefetch when it is available for the pool in use. See “Setting limits for dbcc” on page 622 in the *Performance and Tuning Guide* for more information.

Cache binding commands and the commands to change the size and asynchronous prefetch percentages for pools are dynamic commands. If you use these dbcc commands during off-peak periods, when user applications experience little impact, you can change these settings to speed dbcc performance and restore the normal settings when dbcc checks are finished. See Chapter 19, “Configuring Data Caches,” for information on these commands.

Scheduling database maintenance at your site

There are several factors that determine how often you should run dbcc commands and which ones you need to run.

Database use

If your Adaptive Server is used primarily between the hours of 8:00 a.m. and 5:00 p.m., Monday through Friday, you can run dbcc checks at night and on weekends so that the checks do not have a significant impact on your users. If your tables are not extremely large, you can run a complete set of dbcc commands fairly frequently.

dbcc checkstorage and dbcc checkcatalog provide the best coverage at the lowest cost, assure recovery from backups. You can run dbcc checkdb or dbcc checktable less frequently to check index sort order and consistency. This check does not need to be coordinated with any other database maintenance activity. Reserve object-level dbcc checks and those checks that use the fix option for further diagnosis and correction of faults found by dbcc checkstorage.

If your Adaptive Server is used 24 hours a day, 7 days a week, you may want to limit the resource usage of dbcc checkstorage by limiting the number of worker processes or by using application queues. If you decide not to use dbcc checkstorage, you may want to schedule a cycle of checks on individual tables and indexes using dbcc checktable, dbcc tablealloc, and dbcc indexalloc. At the end of the cycle, when all tables have been checked, you can run dbcc checkcatalog and back up the database. For information on using application queues, see Chapter 4, “Distributing Engine Resources,” in the *Performance and Tuning Guide*.

Some sites with 24-hour, high-performance demands run dbcc checks by:

- Dumping the database to tape
- Loading the database dump into a separate Adaptive Server to create a duplicate database
- Running dbcc commands on the duplicate database
- Running dbcc commands with the fix options on appropriate objects in the original database, if errors are detected that can be repaired with the fix options

The dump is a logical copy of the database pages; therefore, problems found in the original database are present in the duplicate database. This strategy is useful if you are using dumps to provide a duplicate database for reporting or some other purpose.

Schedule the use of dbcc commands that lock objects to avoid interference with business activities. For example, dbcc checkdb acquires locks on all objects in the database while it performs the check. You cannot control the order in which it checks the objects. If you are running an application that uses table4, table5, and table6, and running dbcc checkdb takes 20 minutes to complete, the application will be blocked from accessing these tables, even when the command is not checking them.

Backup schedule

The more often you back up your databases and dump your transaction logs, the more data you can recover in case of failure. You must decide how much data you are willing to lose in the event of a disaster and develop a dump schedule to support that decision.

After you schedule your dumps, decide how to incorporate the dbcc commands into that schedule. You do not have to perform dbcc checks before each dump; however, you may lose additional data if a corruption occurs in the dumps.

An ideal time to dump a database is after you run a complete check of that database using dbcc checkstorage and dbcc checkcatalog. If these commands find no errors in the database, you know that your backup contains a clean database. You can correct problems that occur after loading the dump by reindexing. Use dbcc tablealloc or indexalloc on individual tables and indexes to correct allocation errors reported by dbcc checkalloc.

Size of tables and importance of data

Answer the following questions about your data:

- How many tables contain highly critical data?
- How often does that data change?
- How large are those tables?

dbcc checkstorage is a database-level operation. If only a few tables contain critical data or data that changes often, you may want to run the table- and index-level dbcc commands more frequently on those tables than you run dbcc checkstorage on the entire database.

Understanding the output from *dbcc* commands

dbcc checkstorage stores the results in the dbccdb database. You can print a variety of reports from this database. For details, see “dbcc checkstorage” on page 726.

The output of most other dbcc commands includes information that identifies the objects being checked and error messages that indicate any problems, the command finds in the object. When you run dbcc tablealloc and dbcc indexalloc with fix, the output also indicates the repairs that the command makes.

The following example shows dbcc tablealloc output for a table with an allocation error:

```
dbcc tablealloc(table5)
```

```
Information from sysindexes about the object being checked:
TABLE: table5           OBJID = 144003544
INDID=0  FIRST=337       ROOT=2587       SORT=0

Error message:
Msg 7939, Level 22, State 1:
Line 2:
Table Corrupt: The entry is missing from the OAM for object id
144003544 indid 0 for allocation page 2560.

Message indicating that the error has been corrected:
The missing OAM entry has been inserted.
        Data level: 0.  67 Data  Pages in 9 extents.

dbcc report on page allocation:
TOTAL # of extents = 9
Alloc page 256 (# of extent=1 used pages=8 ref pages=8)
EXTID:560 (Alloc page: 512) is initialized.  Extent follows:
NEXT=0 PREV=0 OBJID=144003544 ALLOC=0xff DEALL=0x0 INDID=0 STATUS=0x0
Alloc page 512 (# of extent=2 used pages=8 ref pages=8)
Page 864 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x1)
Page 865 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x3)
Page 866 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x7)
Page 867 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0xf)
Page 868 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x1f)
Page 869 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x3f)
Page 870 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0x7f)
Page 871 allocated (Alloc page: 768 Extent ID: 864 Alloc mask: 0xff)
Alloc page 768 (# of extent=1 used pages=8 ref pages=8)
Alloc page 1024 (# of extent=1 used pages=8 ref pages=8)
Alloc page 1280 (# of extent=1 used pages=8 ref pages=8)
Alloc page 1536 (# of extent=1 used pages=8 ref pages=8)
Alloc page 1792 (# of extent=1 used pages=8 ref pages=8)
Alloc page 2048 (# of extent=1 used pages=8 ref pages=8)
(Other output deleted.)

Information on resources used:
Statistical information for this run follows:


Total # of pages read = 68
Total # of pages found cache = 68
Total # of physical reads = 0
Total # of saved I/O = 0

Message printed on completion of dbcc command:
```

```
DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
```

Errors generated by database consistency problems

Errors generated by database consistency problems encountered by dbcc checkstorage are documented in the dbcc_types table. Most are in the ranges 5010–5019 and 100,000–100,032. For information on specific errors, see "dbcc_types" on page 82 of the *Adaptive Server Reference Manual*. dbcc checkstorage records two kinds of faults: soft and hard. For information, see "Comparison of soft and hard faults" on page 743.

Errors generated by database consistency problems encountered by dbcc commands other than dbcc checkstorage usually have error numbers from 2500 to 2599 or from 7900 to 7999. These messages, and others that can result from database consistency problems (such as Error 605), may include phrases like "Table Corrupt" or "Extent not within segment."

Some messages indicate severe database consistency problems; others are not so urgent. A few may require help from Sybase Technical Support, but most can be solved by:

- Running dbcc commands that use the fix option
- Following the instructions in the *Troubleshooting Guide*, which contains step-by-step instructions for resolving many database errors found by dbcc

Whatever techniques are required to solve the problems, the solutions are much easier when you find the problem soon after the occurrence of the corruption or inconsistency. Consistency problems can exist on data pages that are not used frequently, such as a table that is updated only monthly. dbcc can find, and often fix, these problems for you.

Comparison of soft and hard faults

When dbcc checkstorage finds a fault in the target database, it is recorded in the dbcc_faults table as either a **soft fault** or a **hard fault**. The following sections describe the two kinds of faults. For more information, see "Verifying faults with dbcc checkverify" on page 745.

Soft faults

A *soft fault* is an inconsistency in Adaptive Server that is usually not persistent. Most soft faults result from temporary inconsistencies in the target database caused by users updates to the database during dbcc checkstorage or when dbcc checkstorage encounters data definition language (DDL) commands. These faults are not repeated when you run the command a second time. You can reclassify soft faults by comparing the results of the two executions of dbcc checkstorage or by running dbcc tablealloc and dbcc checktable after dbcc checkstorage finds soft faults.

If the same soft faults occur in successive executions of dbcc checkstorage, they are "persistent" soft faults, and may indicate a corruption. If you execute dbcc checkstorage in single-user mode, the soft faults reported are "persistent" soft faults. You can resolve these faults by using sp_dbcc_differentialreport or by running dbcc tablealloc and dbcc checktable. If you use the latter two commands, you need to check only the tables or indexes that exhibited the soft faults.

Hard faults

A *hard fault* is a persistent corruption of Adaptive Server that cannot be corrected by restarting Adaptive Server. Not all hard faults are equally severe. For example, each of the following situations cause a hard fault, but the results are different:

- A page that is allocated to a nonexistent table minimally reduces the available disk storage.
- A table with some rows that are unreachable by a scan might return the wrong results.
- A table that is linked to another table causes the query to stop.

Some hard faults can be corrected by simple actions such as truncating the affected table. Others can be corrected only by restoring the database from a backup.

Verifying faults with *dbcc checkverify*

dbcc checkverify examines the results of the most recent checkstorage operation and reclassifies each soft fault as either a hard fault or an insignificant fault. checkverify acts as a second filter to remove spurious faults from the checkstorage results.

How *dbcc checkverify* works

checkverify reads the recorded faults from dbcc_faults and resolves each soft fault through a procedure similar to that used by the checkstorage operation.

Note checkverify locks the table against concurrent updates, which ensures that the soft faults are reclassified correctly. checkverify does not find errors that have occurred since the last run of checkstorage.

checkverify records information in the dbcc_operation_log and dbcc_operation_results tables the same way that checkstorage does. The recorded value of opid is the same as the opid of the last checkstorage operation. checkverify updates the status column in the dbcc_faults table and inserts a row in the dbcc_fault_params table for the faults it processes.

checkverify does not use the scan or text workspaces.

Each fault found by checkstorage is verified by checkverify as one of the following:

- A hard fault classified as such by checkstorage.
- A soft fault reclassified as hard by checkverify because concurrent activity was ruled out as the cause.
- A soft fault confirmed to be soft by checkverify. Some soft faults that appear when there is no concurrent activity in the database do not represent a significant hazard and are not reclassified as hard. A soft fault is not reclassified if it is informational only and not a corruption.
- A soft fault reclassified as insignificant because it can be attributed to concurrent activity or because subsequent activity masked the original inconsistency.

A fault that is assigned code 100011 (text pointer fault) by checkstorage is verified as hard if the text column has a hard fault. If it does not, it is reclassified as soft.

A fault that is assigned code 100016 (page allocated but not linked) by checkstorage is verified as hard if the same fault appears in two successive checkstorage operations. Otherwise, it is reclassified as soft.

When a fault that is assigned code 100035 (spacebits mismatch) by checkstorage is verified as hard, you can repair it by using dbcc checktable.

When checkverify confirms hard faults in your database, follow the same procedures as you did in previous versions of Adaptive Server to correct the faults.

checkverify classifies the following fault codes as soft faults:

- 100020 – check aborted
- 100025 – row count fault
- 100028 – page allocation off current segment

When to use *dbcc checkverify*

You can verify persistent faults by running checkverify anytime after running checkstorage, even after an extended period of hours or days. However, when deciding your schedule, keep in mind that the database state changes over time, and the changes can mask both soft faults and hard faults.

For example, a page that is linked to a table but not allocated is a hard fault. If the table is dropped, the fault is resolved and masked. If the page is allocated to another table, the fault persists but its signature changes. The page now appears to be linked to two different tables. If the page is reallocated to the same table, the fault appears as a corrupt page chain.

Persistent faults that are corrected by a subsequent database change usually do not pose an operational problem. However, detecting and quickly verifying these faults may locate a source of corruption before more serious problems are encountered or before the signature of the original fault changes. For this reason, Sybase recommends that you run checkverify as soon as possible after running dbcc checkstorage.

Note When checkstorage is executed with the target database in single-user mode, there will be no soft faults and no need to execute checkverify.

---

checkverify runs only one time for each execution of checkstorage. However, if checkverify is interrupted and does not complete, you can run it again. The operation resumes from where it was interrupted.

How to use *dbcc checkverify*

The syntax is:

```
dbcc checkverify(dbname)
```

where *dbname* is the database for which you want to verify checkstorage results.

checkverify operates on the results of the last completed checkstorage operation for the specified database only.

When the checkverify operation is complete, Adaptive Server returns the following message:

```
DBCC checkverify for database name, sequence
n completed at date time. n suspect conditions
resolved as faults and n resolved as innocuous.
n checks were aborted.
```

You can run checkverify automatically after running checkstorage by using sp_dbcc_runcheck.

Dropping a damaged database

Use dbcc dbrepair dropdb from the master database to drop a damaged database. No users, including the user running dbrepair, can be using the database when it is dropped.

The syntax for dbcc dbrepair is:

```
dbcc dbrepair (database_name, dropdb )
```

The Transact-SQL command drop database does not work on a database that cannot be recovered or used.

Preparing to use *dbcc checkstorage*

Before you can use dbcc checkstorage, you must configure Adaptive Server and set up the dbccdb database. Table 25-3 summarizes the steps and commands in the order you should use them. Each action is described in detail in the following sections.

Note The examples in this section assume a server that uses 2K logical pages.

Warning! Do not attempt to perform the actions or use the commands in Table 25-3 before you read the information in the referenced section. You must understand the consequences of each action before you make any changes.

Table 25-3: Tasks for preparing to use dbcc checkstorage

For this action	See	Use this command
1. Obtain recommendations for database size, devices (if dbccdb does not exist), workspace sizes, cache size, and the number of worker processes for the target database.	"Planning resources" on page 749 "Planning workspace size" on page 751	use master sp_plan_dbccdb
2. If necessary, adjust the number of worker processes that Adaptive Server uses.	"Configuring worker processes" on page 753	sp_configure number of worker processes memory per worker process

For this action	See	Use this command
3. Create a named cache for dbcc (optional).	“Setting up a named cache for dbcc” on page 754	sp_cacheconfig
4. Configure a your buffer pool.	“Configuring an 8 page I/O buffer pool” on page 755	sp_poolconfig
5. If dbccdb already exists, drop it and all associated devices before creating a new dbccdb database.		drop database
6. Initialize disk devices for the dbccdb data and the log.	“Allocating disk space for dbccdb” on page 756	disk init
7. Create dbccdb on the data disk device.		create database
8. Add disk segments (optional).	“Segments for workspaces” on page 756	use dbccdb sp_addsegment
9. Populate the dbccdb database and install dbcc stored procedures.		`isql -Usa -P -i $SYBASE/scripts/installdbccdb`
10. Create the workspaces.	“dbccdb Workspaces” on page 87	sp_dbcc_createws

11. Update configuration values.	“Updating the dbcc_config table” on page 759	`sp_dbcc_updateconfig`
		`max worker processes`
		`dbcc named cache`
		`scan workspace`
		`text workspace`
		`OAM count threshold`
		`IO error abort`
		`linkage error abort`

Planning resources

Selecting the appropriate device and size for dbccdb is critical to the performance of dbcc checkstorage operations. sp_plan_dbccdb provides configuration recommendations or facts for the specified target database depending on whether dbccdb exists or not. You use this information to configure Adaptive Server and set up the dbccdb database.

Examples of *sp_plan_dbccdb* output

If dbccdb does not exist, sp_plan_dbccdb returns:

- Minimum size for dbccdb
- Devices that are suitable for dbccdb
- Minimum sizes for the scan and text workspaces
- Minimum cache size
- Number of worker processes

The values recommended for the cache size are approximate because the optimum cache size for dbccdb depends on the pattern of the page allocation in the target database. The following example from a server that uses 2K logical pages shows the output of sp_plan_dbccdb for the pubs2 database when dbccdb does not exist:

```
sp_plan_dbccdb pubs2
```

```
Recommended size for dbccdb is 4MB.

Recommended devices for dbccdb are:

Logical Device Name   Device Size   Physical Device Name

sprocdev              28672         /remote/SERV/sprocs_dat
tun_dat               8192          /remote/SERV/tun_dat
tun_log               4096          /remote/SERV/tun_log

Recommended values for workspace size, cache size and process count are:

dbname        scan ws    text ws    cache    process count

pubs2         64K         64K        640K     1
```

If dbccdb already exists, sp_plan_dbccdb returns:

- Minimum size for dbccdb
- Size of existing dbccdb database
- Minimum sizes for the scan and text workspaces
- Minimum cache size
- Number of worker processes

The following example shows the output of sp_plan_dbccdb for the pubs2 database when dbccdb already exists:

```
sp_plan_dbccdb pubs2
```

```
Recommended size for dbccdb database is 23MB (data = 21MB, log = 2MB).

dbccdb database already exists with size 8MB.

Recommended values for workspace size, cache size and process count are:

dbname                              scan ws    text ws    cache    process count

pubs2                                64K        48K        640K     1
```

If you plan to check more than one database, use the name of the largest one for the target database. If you do not provide a target database name, sp_plan_dbccdb returns configuration values for all databases listed in master..sysdatabases, as shown in the following example:

```
sp_plan_dbccdb
Recommended size for dbccdb is 4MB.

dbccdb database already exists with size 8MB.

Recommended values for workspace size, cache size and process count are:

dbname           scan ws    text ws    cache    process count

master            64K        64K        640K     1
tempdb            64K        64K        640K     1
model             64K        64K        640K     1
sybsystemprocs  384K       112K       1280K     2
pubs2             64K        64K        640K     1
pubs3             64K        64K        640K     1
pubtune          160K        96K       1280K     2
sybsecurity       96K        96K       1280K     2
dbccdb           112K        96K      1280K      2
```

For more information, see sp_plan_dbccdb in the *Adaptive Server Reference Manual*.

Planning workspace size

Two workspaces are required for dbccdb: scan and text. Space requirements for the workspaces depend on the size of the largest database that will be checked. To run concurrent dbcc checkstorage operations, you'll need to set up additional workspaces.

Determining the sIze for the largest database to be checked

Different databases can use the same workspaces. Therefore, the workspaces must be large enough to accommodate the largest database with which they will be used.

Note sp_plan_dbccdb suggests workspace sizes – the following details for determining the workspace size are provided for background information only.

Each page in the target database is represented by one 18-byte row in the scan workspace. This workspace should be approximately 1.1 percent of the target database size.

Every non-null text column in the target database inserts a 22-byte row in the text workspace. If there are *n* non-null text columns in the target database, the size of the text workspace must be at least (22 * *n*) bytes. The number of non-null text columns is dynamic, so allocate enough space for the text workspace to accommodate future demands. The minimum space required for the text workspace is 24 pages.

Number of workspaces that can be used concurrently

You can configure dbccdb to run dbcc checkstorage concurrently on multiple databases. This is possible only when the second and subsequent dbcc checkstorage operations have their own dedicated resources. To perform concurrent dbcc checkstorage operations, each operation must have its own scan and text workspaces, worker processes, and reserved cache.

The total space requirement for workspaces depends on whether the user databases are checked serially or concurrently. If dbcc checkstorage operations are run serially, the largest scan and text workspaces can be used for all user databases. If dbcc checkstorage operations are run concurrently, then dbccdb should be set to accommodate the largest workspaces that will be used concurrently. You can determine the workspace sizes by adding the sizes of the largest databases that will be checked concurrently.

For more information, see “dbccdb Workspaces” on page 87.

Configuring Adaptive Server for *dbcc checkstorage*

This section provides information on configuring Adaptive Server for dbcc checkstorage.

Configuring worker processes

The following parameters affect dbcc checkstorage:

- max worker processes – set this parameter with sp_dbcc_updateconfig. It updates the value of max worker processes in the dbcc_config table for each target database.
- number of worker processes – set this configuration parameter with sp_configure. It updates the *server_name.cfg* file.
- memory per worker process – set this configuration parameter with sp_configure. It updates the *server_name.cfg* file.

After changing the value of the sp_configure parameters, you must restart Adaptive Server for the change to take effect. For details, see Chapter 5, "Setting Configuration Parameters."

max worker processes specifies the maximum number of worker processes used by dbcc checkstorage for each target database, while number of worker processes specifies the total number of worker processes supported by Adaptive Server. Worker processes are not dedicated to running dbcc checkstorage operations.

Set the value for number of worker processes high enough to allow for the number of processes specified by max worker processes. A low number of worker processes reduces the performance and resource consumption of dbcc checkstorage. dbcc checkstorage will not use more processes than the number of database devices used by the database. Cache size, CPU performance, and device sizes might suggest a lower worker processes count. If there are not enough worker processes configured for Adaptive Server, dbcc checkstorage will not run.

maximum parallel degree and maximum scan parallel degree have no effect on the parallel functions of dbcc checkstorage. When maximum parallel degree is set to 1, parallelism in dbcc checkstorage is not disabled.

dbcc checkstorage requires multiple processes, so number of worker processes must be set to at least 1 to allow for a parent process and a worker process.

sp_plan_dbccdb recommends values for the number of worker processes, depending on database size, number of devices, and other factors. You can use smaller values to limit the load on your system. dbcc checkstorage may use fewer worker processes than sp_plan_dbccdb recommends or fewer than you configure.

Using more worker processes does not guarantee faster performance. The following scenario describes the effects of two different configurations:

An 8GB database has 4GB of data on disk A and 0.5GB of data on each of the disks B, C, D, E, F, G, H, and I.

With 9 worker processes active, the time it takes to run dbcc checkstorage is 2 hours, which is the time it takes to check disk A. Each of the other 8 worker processes finishes in 15 minutes and waits for the disk A worker process to finish.

With 2 worker processes active, the time it takes to run dbcc checkstorage is still 2 hours. The first worker process processes disk A and the other worker process processes disks B, C, D, E, F, G, H, and I. In this case, there is no waiting, and resources are used more efficiently.

memory per worker process specifies the total memory allocation for worker processes support in Adaptive Server. The default value is adequate for dbcc checkstorage.

Setting up a named cache for *dbcc*

If you use a named cache for dbcc checkstorage, you might need to adjust the Adaptive Server configuration parameters.

During a dbcc checkstorage operation, the workspaces are temporarily bound to a cache which is also used to read the target database. Using a named cache that is dedicated to dbcc minimizes the impact of the database check on other users and improves performance. You can create a separate cache for each dbcc checkstorage operation that will be run concurrently, or you can create one cache that is large enough to fit the total requirements of the concurrent operations. The size required for optimum performance depends on the size of the target database and distributions of data in that database. dbcc checkstorage requires a minimum of 640K of buffers (each buffer is one extent) per worker process in the named cache.

For best performance, assign most of the dedicated cache to the buffer pool and do not partition the cache. The recommended cache size is the minimum size for the bufffer pool. Add the size of the one page pool to this value.

If you dedicate a cache for dbcc checkstorage, the command does not require more than the minimum one page buffer pool. If the cache is shared, you can improve the performance of dbcc checkstorage by increasing the buffer pool size before running the operation, and reducing the size after the operation is complete. The buffer pool requirements are the same for a shared cache. However, while a shared cache may meet the size requirement, other demands on the cache might limit the buffer availability to dbcc checkstorage and greatly impact the performance of both checkstorage and Adaptive Server as a whole.

Warning! Do not use cache partitions in a cache being used for dbcc checkstorage.

To configure Adaptive Server with a named cache for dbcc checkstorage operations, use sp_cacheconfig and sp_poolconfig. See Chapter 19, “Configuring Data Caches.”

Configuring an 8 page I/O buffer pool

dbcc checkstorage requires a I/O buffer pool of one extent. Use sp_poolconfig to configure the pool size and verify that the pool has been configured properly. The pool size is stored in the dbcc_config table.

The following example shows how to use sp_poolconfig to set a 16K buffer pool for “master_cache” on a server configured for 2K logical pages. The named cache that is created for the master database.

```
1> sp_poolconfig "master_cache", "1024K", "16K"
2> go
(return status = 0)
```

The following example shows that the buffer pool for the private cache “master_cache” is set:

```
1> sp_poolconfig "master_cache"
2> go
```

```
Cache Name     Status    Type     Config Value Run Value
-------------  --------- -------- ------------ -----------
master_cache   Active    Mixed         2.00 Mb      2.00 Mb
```

```
                                 ------------ ------------
                         Total        2.00 Mb      2.00 Mb
=================================================================
Cache: master_cache,   Status: Active,   Type: Mixed
   Config Size: 2.00 Mb,   Run Size: 2.00 Mb
   Config Replacement: strict LRU,   Run Replacement: strict LRU
IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
2 Kb     512 Kb    0.00 Mb      1.00 Mb      10
16 Kb    192 Kb    1.00 Mb      1.00 Mb      10
(return status = 0)
```

For more information on sp_poolconfig, see the Adaptive Server Reference Manual.

Allocating disk space for *dbccdb*

Additional disk storage is required for the dbccdb database. Because dbcc checkstorage uses dbccdb extensively, you should place dbccdb on a device that is separate from other database devices.

Note Do not create dbccdb on the master device. Make sure that the log devices and data devices for dbccdb are separate.

Segments for workspaces

By dedicating segments for workspaces, you can control the workspace placement and improve the performance of dbcc checkstorage performance. When you dedicate new segments for the exclusive use of workspaces, be sure to unmap the devices attached to these segments from the default segment with sp_dropsegment.

Creating the *dbccdb* database

To create the dbccdb database:

1 Run sp_plan_dbccdb in the master database to obtain recommendations for database size, devices, workspace sizes, cache size, and the number of worker processes for the target database. For example, suppose you run sp_plan_dbccdb with pubs2 as the target database when dbccdb did not exist:

```
use master
go
sp_plan_dbccdb pubs2
go
```

The following output appears:

```
Recommended size for dbccdb is 4MB.

Recommended devices for dbccdb are:

Logical Device Name   Device Size   Physical Device Name

sprocdev              28672         /remote/SERV/sprocs_dat
tun_dat               8192          /remote/SERV/tun_dat
tun_log               4096          /remote/SERV/tun_log

Recommended values for workspace size, cache size and process count are:

dbname        scan ws    text ws    cache    process count

pubs2         64K         64K        640K     1
```

For details on the information provided by sp_plan_dbccdb, see "Planning resources" on page 749.

2. If dbccdb already exists, drop it and all associated devices before creating a new dbccdb database:

```
use master
go
if exists (select * from master.dbo.sysdatabases
     where name = "dbccdb")
begin
     print "+++ Dropping the dbccdb database"
     drop database dbccdb
end
go
```

3. Use disk init to initialize disk devices for the dbccdb data and the log:

```
use master
go
disk init
   name = "dbccdb_dat",
   physname = "/remote/disks/masters/",
   size = “4096K”
go
disk init
```

```
   name = "dbccdb_log",
   physname = "/remote/disks/masters/",
size = “1024K”
go
```

4 Use create database to create dbccdb on the data disk device that you initialized in step 3:

```
use master
go

create database dbccdb
   on dbccdb_dat = 6
   log on dbccdb_log = 2
go
```

5 (Optional) – add segments for the scan and text workspaces to the dbccdb data device:

```
use dbccdb
go
sp_addsegment scanseg, dbccdb, dbccdb_dat
go
sp_addsegment textseg, dbccdb, dbccdb_dat
go
```

6 Create the tables for dbccdb and initialize the dbcc_types table:

```
isql -Ujms -P***** -iinstalldbccdb
```

The installdbccdb script checks for the existence of the database before it attempts to create the tables. It creates only those tables that do not already exist in dbccdb. If any of the dbccdb tables become corrupted, remove them with drop table, and then use installdbccdb to re-create them.

7 Create and initialize the scan and text workspaces:

```
use dbccdb
go
sp_dbcc_createws dbccdb, scanseg, scan_pubs2,
scan, "10M"
go
sp_dbcc_createws dbccdb, textseg, text_pubs2,
text, "10M"
go
```

When you have finished installing dbccdb, you must update the dbcc_config table.

Updating the *dbcc_config* table

Use sp_dbcc_updateconfig to initialize the dbcc_config table for the *target database*. You must update each dbcc parameter separately for each target database, as shown in the following example.

```
use dbccdb
go

sp_dbcc_updateconfig pubs2,"max worker processes", "4"
go

sp_dbcc_updateconfig pubs2, "dbcc named cache", pubs2_cache, "10K"
go

sp_dbcc_updateconfig pubs2, "scan workspace", scan_pubs2
go

sp_dbcc_updateconfig pubs2, "text workspace", text_pubs2
go

sp_dbcc_updateconfig pubs2, "OAM count threshold", "5"
go

sp_dbcc_updateconfig pubs2, "IO error abort", "3"
go

sp_dbcc_updateconfig pubs2,"linkage error abort", "8"
go
```

You can now use dbcc checkstorage to check your databases. For descriptions of the dbcc parameters, see type code values 1 through 9 in “dbcc_types” on page 82.

Maintaining *dbccdb*

You will occasionally need to perform maintenance tasks on dbccdb.

- Reevaluate and update the configuration using:
    - sp_dbcc_evaluatedb – recommends values for configuration parameters using the results of previous dbcc checkstorage operations.

    - sp_dbcc_updateconfig – updates the configuration parameters for the specified database.
- Clean up obsolete data in dbccdb:
    - sp_dbcc_deletedb – deletes all the information on the specified database from dbccdb.
    - sp_dbcc_deletehistory – deletes the results of the dbcc checkstorage operations on the specified database from dbccdb.
- Remove unnecessary workspaces.
- Perform consistency checks on dbccdb itself.

The following sections describe the maintenance tasks in greater detail.

Reevaluating and updating *dbccdb* configuration

If the characteristics of user databases change, use sp_dbcc_evaluatedb to reevaluate the current dbccdb configuration and recommend more suitable values.

The following changes to user databases might affect the dbccdb configuration, as follows:

- When a user database is created, deleted or altered, the size of the workspaces and named cache, or the number of worker threads stored in the dbcc_config table might be affected.
- Changes in the named cache size or worker process count for dbcc_checkstorage may require you to reconfigure buffer cache and worker processes.

If the results of dbcc checkstorage operations are available for the target database, use sp_dbcc_evaluatedb to determine new configuration values. sp_dbcc_configreport also reports the configuration parameters for the specified database.

Use sp_dbcc_updateconfig to add new databases to the dbcc_config table and to change the configuration values in dbcc_config to reflect the values recommended by sp_dbcc_evaluatedb.

Cleaning up *dbccdb*

Adaptive Server stores data generated by dbcc checkstorage in dbccdb. You should periodically clean up dbccdb by using sp_dbcc_deletehistory to delete data for the target database that was created before the date you specify.

When you delete a database, you should also delete from dbccdb all configuration information and dbcc checkstorage results related to that database. Use sp_dbcc_deletedb to delete all database information from dbccdb.

Removing workspaces

You may need to remove unnecessary workspaces. In dbccdb, issue:

```
drop table workspace_name
```

Performing consistency checks on *dbccdb*

The limited update activity in the dbccdb tables should make corruption less frequent. Two signs of corruption in dbccdb are:

- Failure of dbcc checkstorage during the initialization phase, as it evaluates the work that needs to be performed, or during the completion phase, when it records its results
- Loss of information about faults resulting from corruption in the recorded faults, found by dbcc checkstorage

A severe corruption in dbccdb may cause dbcc checkstorage to fail. For dbcc checkstorage to locate severe corruptions in dbccdb, you can create an alternate database, dbccalt, which you use only for checking dbccdb. Create dbccalt using the same process that you used to create dbccdb as described in “Preparing to use dbcc checkstorage” on page 748.

If no free devices are available for dbccalt, you can use any device that is not used by the master database or dbccdb.

dbcc checkstorage and the dbcc system procedures function the same with dbccalt as they do with dbccdb. When the target database is dbccdb, dbcc checkstorage uses dbccalt, if it exists. If dbccalt does not exist, dbccdb can be checked using itself as the management database. If the target database is dbccdb and dbccalt exists, the results of dbcc checkstorage operations on dbccdb are stored in dbccalt. If dbccalt does not exist, the results are stored in dbccdb itself.

Alternatively, dbcc checkalloc and dbcc checktable can be used to check dbccdb.

If dbccdb becomes corrupted, you can drop it and re-create it or load an older version from a backup. If you drop it, some of its diagnostic history will be lost.

Generating reports from *dbccdb*

Several dbcc stored procedures are provided with dbccdb so that you can generate reports from the data in dbccdb.

To report a summary of *dbcc checkstorage* operations

sp_dbcc_summaryreport reports all dbcc checkstorage operations that were completed for the specified database on or before the specified date. The following example shows output from this command:

```
                           sp_dbcc_summaryreport
DBCC Operation : checkstorage

Database Name      Start time          End Time   Operation ID
         Hard Faults Soft Faults Text Columns Abort Count
         User Name
------------------ ------------------- ---------- -----------
         ----------- ----------- ------------ -----------
         ------------------------------
sybsystemprocs     05/12/1997 10:54:45  10:54:53            1
        0           0            0           0
     sa
sybsystemprocs     05/12/1997 11:14:10  11:14:19            2
        0           0            0           0
     sa
```

For details, see Chapter 27, “dbcc Stored Procedures,” in the *Adaptive Server Reference Manual*.

To report configuration, statistics and fault information

sp_dbcc_fullreport runs these reports in the order shown:

- sp_dbcc_summaryreport – for an example, see “To report a summary of dbcc checkstorage operations” on page 762.
- sp_dbcc_configreport – for an example, see “To see configuration information for a target database” on page 763.
- sp_dbcc_statisticsreport – for an example, see “To report statistics information from dbcc_counter” on page 765.
- sp_dbcc_faultreport short – for an example, see “To report faults found in a database object” on page 764.

To see configuration information for a target database

Use sp_dbcc_configreport to generate a report of the configuration information for a target database. The following example shows output from this command:

```
sp_dbcc_configreport
```

```
Reporting configuration information of database sybsystemprocs.

Parameter Name            Value                         Size

database name             sybsystemprocs                51200K
dbcc named cache          default data cache            1024K
text workspace            textws_001 (id = 544004969)   128K
scan workspace            scanws_001 (id = 512004855)   1024K
max worker processes      1
operation sequence number 2
```

To compare results of *dbcc checkstorage* operations

sp_dbcc_differentialreport compares the results of the dbcc checkstorage operations completed for the specified database object on the specified dates. The following example shows output from this command:

```
sp_dbcc_differentialreport master, sysprocedures,
checkstorage, "01/01/96", "01/02/96"
```

```
The following changes in dbcc counter values for the object "sysprocedures" in
database master have been noticed between 01/01/96 and 01/02/96.

Description        Date1   Date2

pages used           999    1020
pages reserved      1000    1024
page extent gaps      64      67
```

To report faults found in a database object

sp_dbcc_faultreport reports faults in the specified database object that occurred on or before the specified date. You can generate a short or long report. The following example shows a short report:

```
sp_dbcc_faultreport ’short’
```

```
Database Name : sybsystemprocs

 Table Name     Index  Type Code Description        Page Number
 -------------- ------ --------- ------------------ -----------
 sysprocedures       0    100031 page not allocated        5702
 sysprocedures       1    100031 page not allocated       14151
 syslogs             0    100022 chain start error        24315
 syslogs             0    100031 page not allocated       24315
```

The following example shows part of the output of a long report for the sybsystemprocs database. The complete report repeats the information for each object in the target database.

```
sp_dbcc_faultreport ’long’
```

```
Generating ’Fault Report’ for object sysprocedures in database sybsystemprocs.

Type Code: 100031; Soft fault, possibly spurious
Page reached by the chain is not allocated.
page id:  14151
page header:
0x0000374700003788000037460000005000648B803EF0001000103FE0080000F
Header for 14151, next 14216, previous 14150, id = 5:1
 time stamp = 0x0001000648B8, next row = 1007, level = 0
 free offset = 1022, minlen = 15, status = 128(0x0080)
.
.
```

To report statistics information from *dbcc_counter*

sp_dbcc_statisticsreport reports statistics information from the dbcc_counter table generated by dbcc checkstorage on or before the specified date. The following example shows output from this command:

```
sp_dbcc_statisticsreport 'sybsystemprocs',
'sysobjects'
```

```
Statistics Report on object sysobjects in database sybsystemprocs

 Parameter Name    Index Id Value
 ----------------- -------- -----------
 count             0        160.0
 max size          0        99.0
 max count         0        16.0
 bytes data        0        12829.0
 bytes used        0        15228.0
 count             1        16.0
 max size          1        9.0
 max level         1        0.0
 max count         1        16.0
 bytes data        1        64.0
 bytes used        1        176.0
 count             2        166.0
 max level         2        1.0
 max size          2        39.0
 max count         2        48.0
 bytes data        2        3092.0
 bytes used        2        4988.0


 Parameter Name         Index Id Partition Value    Dev_name
 ---------------------- -------- --------- -------- ----------------
 page gaps              0        1         16.0     master
 pages used             0        1         17.0     master
 extents used           0        1         3.0      master
```

```
overflow pages         0        1         0.0      master
pages overhead         0        1         1.0      master
pages reserved         0        1         6.0      master
page extent gaps       0        1         7.0      master
ws buffer crosses      0        1         7.0      master
page extent crosses    0        1         7.0      master
page gaps              1        1         1.0      master
pages used             1        1         2.0      master
extents used           1        1         1.0      master
overflow pages         1        1         0.0      master
pages overhead         1        1         1.0      master
pages reserved         1        1         6.0      master
page extent gaps       1        1         0.0      master
ws buffer crosses      1        1         0.0      master
page extent crosses    1        1         0.0      master
page gaps              2        1         5.0      master
pages used             2        1         8.0      master
extents used           2        1         1.0      master
overflow pages         2        1         0.0      master
pages overhead         2        1         1.0      master
pages reserved         2        1         0.0      master
page extent gaps       2        1         0.0      master
ws buffer crosses      2        1         0.0      master
page extent crosses    2        1         0.0      master
```

CHAPTER 26

Developing a Backup and Recovery Plan

Adaptive Server has **automatic recovery** procedures that protect you from power outages and computer failures. To protect yourself against media failure, you must make regular and frequent backups of your databases.

This chapter provides information to help you develop a backup and recovery plan. The first part of this chapter provides an overview of Adaptive Server's backup and recovery processes. The second part of this chapter discusses the backup and recovery issues that you should address before you begin using your system for production.

Keeping track of database changes

Adaptive Server uses transactions to keep track of all database changes. Transactions are Adaptive Server's units of work. A transaction consists of one or more Transact-SQL statements that succeed—or fail—as a unit.

Each SQL statement that modifies data is considered a **transaction**. Users can also define transactions by enclosing a series of statements within a begin transaction...end transaction block. For more information about transactions, seeChapter 18, "Transactions: Maintaining Data Consistency and Recovery," in the *Transact-SQL User's Guide*.

Each database has its own **transaction log**, the system table syslogs. The transaction log automatically records every transaction issued by each user of the database. You cannot turn off transaction logging.

The transaction log is a *write-ahead log*. When a user issues a statement that will modify the database, Adaptive Server writes the changes to the log. After all changes for a statement have been recorded in the log, they are written to an in-cache copy of the data page. The data page remains in cache until the memory is needed for another database page. At that time, it is written to disk.

If any statement in a transaction fails to complete, Adaptive Server reverses all changes made by the transaction. Adaptive Server writes an "end transaction" record to the log at the end of each transaction, recording the status (success or failure) of the transaction.

Getting information about the transaction log

The transaction log contains enough information about each transaction to ensure that it can be recovered. Use the dump transaction command to copy the information it contains to tape or disk. Use sp_spaceused syslogs to check the size of the log, or sp_helpsegment logsegment to check the space available for log growth.

Warning! Never use insert, update, or delete commands to modify syslogs.

Synchronizing a database and its log: checkpoints

A checkpoint writes all dirty pages—pages that have been modified in memory, but not on disk, since the last checkpoint—to the database device. Adaptive Server's automatic **checkpoint** mechanism guarantees that data pages changed by completed transactions are regularly written from the memory cache to the database device. Synchronizing the database and its transaction log shortens the time it takes to recover the database after a system crash.

Setting the recovery interval

Typically, automatic recovery takes from a few seconds to a few minutes per database. The time varies, depending on the size of the database, the size of the transaction log, and the number and size of the transactions that must be committed or rolled back.

Use sp_configure with the recovery interval in minutes parameter to specify, the maximum permissible recovery time. Adaptive Server runs automatic checkpoints often enough to recover the database within that period of time.

```
sp_configure "recovery interval in minutes"
```

The default value, 5, allows recovery within 5 minutes per database. To change the recovery interval to 3 minutes, use:

```
sp_configure "recovery interval in minutes", 3
```

Note The recovery interval has no effect on long-running, minimally logged transactions (such as create index) that are active at the time Adaptive Server fails. It may take as much time to reverse these transactions as it took to run them. To avoid lengthy delays, dump each database immediately after you create an index on one of its tables.

Automatic checkpoint procedure

Approximately once a minute, the checkpoint task checks each database on the server to see how many records have been added to the transaction log since the last checkpoint. If the server estimates that the time required to recover these transactions is greater than the database's recovery interval, Adaptive Server issues a checkpoint.

The modified pages are written from cache onto the database devices, and the checkpoint event is recorded in the transaction log. Then, the checkpoint task "sleeps" for another minute.

To see the checkpoint task, execute sp_who. The checkpoint task is usually displayed as "CHECKPOINT SLEEP" in the "cmd" column:

```
spid  status     loginame   hostname blk dbname  cmd
----- ---------- ---------- -------- --- ------- ----------------
    1 running    sa         mars     0   master  SELECT
    2 sleeping   NULL                0   master  NETWORK HANDLER
    3 sleeping   NULL                0   master  MIRROR HANDLER
    4 sleeping   NULL                0   master  HOUSEKEEPER
    5 sleeping   NULL                0   master  CHECKPOINT SLEEP
```

Checkpoint after user database upgrade

Adaptive Server inserts a checkpoint record immediately after upgrading a user database. Adaptive Server uses this record to ensure that a dump database occurs before a dump transaction occurs on the upgraded database.

Truncating the log after automatic checkpoints

System Administrators can truncate the transaction log when Adaptive Server performs an automatic checkpoint.

To set the trunc log on chkpt database option, which will truncate the transaction log if it consists of 50 or more rows when an automatic checkpoint occurs, execute this command from the master database:

```
sp_dboption database_name, "trunc log on chkpt", true
```

This option is not suitable for production environments because it does not make a copy of the transaction log before truncating it. Use trunc log on chkpt only for:

- Databases whose transaction logs cannot be backed up because they are not on a separate segment
- Test databases for which current backups are not important

Note If you leave the trunc log on chkpt option set to off (the default condition), the transaction log continues to grow until you truncate it with the dump transaction command.

To protect your log from running out of space, you should design your last-chance threshold procedure to dump the transaction log. For more information about threshold procedures, see Chapter 29, “Managing Free Space with Thresholds.”

Free checkpoints

When Adaptive Server has no user tasks to process, a housekeeper task automatically begins writing dirty buffers to disk. If the housekeeper task is able to flush all active buffer pools in all configured caches, it wakes up the checkpoint task. The checkpoint task determines whether it needs to perform a checkpoint on the database.

Checkpoints that occur as a result of the housekeeper task are known as *free checkpoints*. They do not involve writing many dirty pages to the database device, since the housekeeper task has already done this work. They may result in a shorter recovery time for the database.

For information about tuning the housekeeper task, see Chapter 3, “Using Engines and CPUs,” in the *Performance and Tuning Guide*.

Manually requesting a checkpoint

Database Owners can issue the checkpoint command to force all modified pages in memory to be written to disk. Manual checkpoints do not truncate the log, even if the trunc log on chkpt option of sp_dboption is turned on.

Use the checkpoint command:

- As a precautionary measure in special circumstances—for example, just before a planned shutdown with nowait so that Adaptive Server’s recovery mechanisms will occur within the recovery interval. (An ordinary shutdown performs a checkpoint.)

- To cause a change in database options to take effect after executing the sp_dboption system procedure. (After you run sp_dboption, an informational message reminds you to run checkpoint.)

Automatic recovery after a system failure or shutdown

Each time you restart Adaptive Server—for example, after a power failure, an operating system failure, or the use of the shutdown command—it automatically performs a set of recovery procedures on each database.

The recovery mechanism compares each database to its transaction log. If the log record for a particular change is more recent than the data page, the recovery mechanism reapplies the change from the transaction log. If a transaction was ongoing at the time of the failure, the recovery mechanism reverses all changes that were made by the transaction.

When you boot Adaptive Server, it performs database recovery in this order:

1. Recovers master
2. Recovers sybsystemprocs
3. Recovers model
4. Creates tempdb (by copying model)
5. Recovers sybsystemdb
6. Recovers sybsecurity
7. Recovers user databases, in order by sysdatabases.dbid, or according to the order specified by sp_dbrecovery_order. See below for more information about sp_dbrecovery_order.

Users can log in to Adaptive Server as soon as the system databases have been recovered, but they cannot access other databases until they have been recovered.

Determining whether messages are displayed during recovery

The configuration variable print recovery information determines whether Adaptive Server displays detailed messages about each transaction on the console screen during recovery. By default, these messages are not displayed. To display messages, use:

```
sp_configure "print recovery information", 1
```

User-defined database recovery order

sp_dbrecovery_order allows you to determine the order in which individual user databases recover. This makes it possible to assign a recovery order in which, for example, critical databases recover before lower-priority databases.

Important features of recovery order are:

- System databases are recovered first, in this order:
  - master
  - model
  - tempdb
  - sybsystemdb
  - sybsecurity
  - sybsystemprocs

  All other databases are considered user databases, and you can specify their recovery order.
- You can use sp_dbrecovery_order to specify the recovery order of user-databases and to list the user-defined recovery order of an individual database or of all databases.
- User databases that are not explicitly assigned a recovery order with sp_dbrecovery_order are recovered according to their database ID, after all the databases that have a user-defined recovery order.
- If you do not use sp_dbrecovery_order to assign any databases a recovery order, user databases are recovered in order of database ID.

Using *sp_dbrecovery_order*

To use sp_dbrecovery_order to enter or modify a user-defined recovery order, you must be in the master database and have System Administrator privileges. Any user, in any database, can use sp_dbrecovery_order to list the user-defined recovery order of databases.

The syntax for sp_dbrecovery_order is:

```
sp_dbrecovery_order
    [database_name [, rec_order [, force]]]
```

where *database_name* is the name of the user database to which you want to assign a recovery order, and *rec_order* is the order in which the database is to be recovered.

Recovery order must be consecutive, starting with 1. You cannot assign a recovery sequence of 1, 2, 4, with the intention of assigning a recovery order of 3 to another database at a later time.

To insert a database into a user-defined recovery sequence without putting it at the end, enter *rec_order* and specify force. For example, if databases A, B, and C have a user-defined recovery order of 1, 2, 3, and you want to insert the pubs2 database as the second user database to recover, enter:

```
sp_dbrecovery_order pubs2, 2, force
```

This command assigns a recovery order of 3 to database B and a recovery order of 4 to database C.

Changing or deleting the recovery position of a database

To change the position of a database in a user-defined recovery sequence, delete the database from the recovery sequence and then insert it in the position you want it to occupy. If the new position is not at the end of the recovery order, use the force option.

To delete a database from a recovery sequence, specify a recovery order of -1.

For example, to move the pubs2 database from recovery position 2 to recovery position 1, delete the database from the recovery sequence and then reassign it a recovery order as follows:

```
sp_dbrecovery_order pubs2, -1
sp_dbrecovery_order pubs2, 1, "force"
```

Listing the user-assigned recovery order of databases

To list the recovery order of all databases assigned a recovery order, use:

```
sp_dbrecovery_order
```

This generates output similar to:

```
The following databases have user specified recovery order:
Recovery Order Database Name        Database Id
-------------- -------------------- -----------
    1          dbccdb                   8
    2          pubs2                    5
    3          pubs3                    6
    4          pubtune                  7
The rest of the databases will be recovered in default database id order.
```

To display the recovery order of a specific database, enter the database name:

```
1> sp_dbrecovery_order pubs2
2> go
Database Name Database id Recovery Order
 ------------ ----------- --------------
 pubs2             5          2
```

Fault isolation during recovery

The recovery procedures, known simply as “recovery,” rebuild the server’s databases from the transaction logs. The following situations cause recovery to run:

- Adaptive Server start-up
- Use of the load database command
- Use of the load transaction command

The recovery isolation mode setting controls how recovery behaves when it detects corrupt data while reversing or reapplying a transaction in a database.

If an index is marked as suspect, the System Administrator can repair this by dropping and re-creating the index.

Recovery fault isolation provides the ability to:

- Configure whether an entire database or just the suspect pages become inaccessible when recovery detects corruption
- Configure whether an entire database with suspect pages comes online in read_only mode or whether the online pages are accessible for modification
- List databases that have suspect pages
- List the suspect pages in a specified database by page ID, index ID, and object name
- Bring suspect pages online for the System Administrator while they are being repaired
- Bring suspect pages online for all database users after they have been repaired

The ability to isolate only the suspect pages while bringing the rest of the database online provides a greater degree of flexibility in dealing with data corruption. You can diagnose problems, and sometimes correct them, while most of the database is accessible to users. You can assess the extent of the damage and schedule emergency repairs or reload for a convenient time.

Recovery fault isolation applies only to user databases. Recovery always takes a system database entirely offline if it has any corrupt pages. You cannot recover a system database until you have repaired or removed all of its corrupt pages.

Persistence of offline pages

Suspect pages that you have taken offline remain offline when you reboot the server. Information about offline pages is stored in master.dbo.sysattributes.

Use the drop database and load database commands to clear entries for suspect pages from master.dbo.sysattributes.

Configuring recovery fault isolation

When Adaptive Server is installed, the default recovery isolation mode is “databases,” which marks a database as suspect and takes the entire database offline if it detects any corrupt pages.

Isolating suspect pages

To isolate the suspect pages so that only they are taken offline, while the rest of the database remains accessible to users, use the sp_setsuspect_granularity to set the recovery isolation mode to "page." This mode will be in effect the next time that recovery is performed in the database.

The syntax for sp_setsuspect_granularity is:

```
sp_setsuspect_granularity
    [dbname [,{"database" | "page"} [, "read_only"]]]
```

With the *dbname* and either database or page as the second argument, sp_setsuspect_granularity sets the recovery isolation mode.

Without the database or page argument, sp_setsuspect_granularity displays the current and configured recovery isolation mode settings for the specified database. Without any arguments, it displays those settings for the current database.

If corruption cannot be isolated to a specific page, recovery marks the entire database as suspect, even if you set the recovery isolation mode to "page." For example, a corrupt transaction log or the unavailability of a global resource causes this to occur.

When recovery marks specific pages as suspect, the default behavior is for the database to be accessible for reading and writing with the suspect pages offline and therefore inaccessible. However, if you specify the read_only option to sp_setsuspect_granularity, and recovery marks any pages as suspect, the entire database comes online in read_only mode and cannot be modified. If you prefer the read_only option, but in certain cases you are comfortable allowing users to modify the non-suspect pages, you can make the online portion of the database writable with sp_dboption:

```
sp_dboption pubs2, "read only", false
```

In this case, the suspect pages remain offline until you repair them or force them, as described in "Bringing offline pages online" on page 779.

Raising the number of suspect pages allowed

The suspect escalation threshold is the number of suspect pages at which recovery marks an entire database suspect, even if the recovery isolation mode is "page." By default, it is set to 20 pages in a single database. You can use sp_setsuspect_threshold to change the suspect escalation threshold.

The syntax for sp_setsuspect_threshold is:

```
sp_setsuspect_threshold [dbname [,threshold]]
```

With the *dbname* and *threshold* arguments, sp_setsuspect_threshold displays the current and configured suspect escalation threshold settings for the specified database. Without any arguments, it displays these settings for the current database.

You configure recovery fault isolation and the suspect escalation threshold at the database level.

You cannot execute sp_setsuspect_granularity or sp_setsuspect_threshold inside a transaction.

You must have the sa_role and be in the master database to set values with sp_setsuspect_granularity and sp_setsuspect_threshold. Any user can execute these procedures with only the name of the database as an argument to display the values configured for that database, as illustrated below:

```
sp_setsuspect_granularity pubs2
DB Name  Cur. Suspect Gran.  Cfg. Suspect Gran.  Online mode
-------  ------------------  ------------------  -----------
pubs2    page                page                read/write
sp_setsuspect_threshold pubs2
DB Name       Cur. Suspect threshold   Cfg. Suspect threshold
------------- ------------------------ ----------------------
pubs2         20                       30
```

This example shows that the recovery isolation mode for the pubs2 database was “page” and the escalation threshold was 20 the last time recovery ran on this database (the current suspect threshold values). The next time recovery runs on this database, the recovery isolation mode will be “page” and the escalation threshold will be 30 (the configured values).

With no arguments, sp_setsuspect_granularity and sp_setsuspect_threshold display the current and configured settings for the current database, if it is a user database.

Getting information about offline databases and pages

To see which databases have offline pages, use sp_listsuspect_db. The syntax is:

```
sp_listsuspect_db
```

The following example displays general information about the suspect pages:

```
sp_listsuspect_db
The database ’dbt1’ has 3 suspect pages belonging to 2 objects.
(return status = 0)
```

To get detailed information about the individual offline pages, use sp_listsuspect_page. The syntax is:

```
sp_listsuspect_page [dbname]
```

If you don’t specify the dbname, the defaults is the current database. The following example shows the detailed page-level output of sp_listsuspect_page in the dbt1 database.

```
sp_listsuspect_page dbt1
DBName   Pageid       Object    Index   Access
-------- ------------ --------- ------- -------------------
dbt1     384          tab1      0       BLOCK_ALL
dbt1     390          tab1      0       BLOCK_ALL
dbt1     416          tab1      1       SA_ONLY

(3 rows affected, return status = 0)
```

If the value in the “Access” column is SA_ONLY, the suspect page is 1, the suspect page is accessible to users with the sa_role. If it is BLOCK_ALL, no one can access the page.

Any user can run sp_listsuspect_db and sp_listsuspect_page from any database.

Bringing offline pages online

To make all the offline pages in a database accessible, use sp_forceonline_db. The syntax is:

```
sp_forceonline_db dbname,
    {"sa_on" | "sa_off" | "all_users"}
```

To make an individual offline page accessible, use sp_forceonline_page. The syntax is:

```
sp_forceonline_page dbname, pgid
    {"sa_on" | "sa_off" | "all_users"}
```

With both of these procedures, you specify the type of access.

- "sa_on" makes the suspect page or database accessible only to users with the sa_role. This is useful for repairing the suspect pages and testing the repairs while the database is up and running, without allowing normal users access to the suspect pages. You can also use it to perform a dump database or a dump transaction with no_log on a database with suspect pages, which would be prohibited if the pages were offline.
- "sa_off" blocks access to all users, including System Administrators. This reverses a previous sp_forceonline_db or sp_forceonline_page with "sa_on."
- "all_users" brings offline pages online for all users after the pages have been repaired.

  Unlike bringing suspect pages online with "sa_on" and then making them offline again with "sa_off," when you use sp_forceonline_page or sp_forceonline_db to bring pages online for "all users," this action cannot be reversed. There is no way to make the online pages offline again.

Warning! Adaptive Server does not perform any checks on pages being brought online. It is your responsibility to ensure that pages being brought online have been repaired.

You cannot execute sp_forceonline_db or sp_forceonline_page inside a transaction.

You must have the sa_role and be in the master database to execute sp_forceonline_db and sp_forceonline_page.

Index-level fault isolation for data-only-locked tables

When pages of an index for a data-only-locked table are marked as suspect during recovery, the entire index is taken offline. Two system procedures manage offline indexes:

- sp_listsuspect_object
- sp_forceonline_object

In most cases, a System Administrator uses sp_forceonline_object to make a suspect index available only to those with the sa_role. If the index is on a user table, you can repair the suspect index by dropping and re-creating the index.

See the Adaptive Server Reference Manual for more information about sp_listsuspect_objec and sp_forceonline_object.

Side effects of offline pages

The following restrictions apply to databases with offline pages:

- Transactions that need offline data, either directly or indirectly (for example, because of referential integrity constraints), fail and generate a message.
- You cannot use dump database when any part of the database is offline.

  A System Administrator can force the offline pages online using sp_forceonline_db with "sa_on," dump the database, and then use sp_forceonline_db with "sa_off" after the dump completes.
- You cannot use dump transaction with no_log or dump transaction with truncate_only if any part of a database is offline.

  A System Administrator can force the offline pages online using sp_forceonline_db with "sa_on,", dump the transaction log using with no_log, and then use sp_forceonline_db with "sa_off" after the dump completes.
- If you want to drop a table or index containing offline pages, you must use a transaction in the master database. Otherwise, the drop will fail because it needs to delete entries for the suspect pages from master.dbo.sysattributes. The following example drops the object and deletes information about its offline pages from master.dbo.sysattributes.

  To drop an index named authors_au_id_ind, which contains suspect pages, from the pubs2 database, drop the index inside a master database transaction as follows:

  ```
  use master
  go
  sp_dboption pubs2, "ddl in tran", true
  go
  ```

```
use pubs2
go
checkpoint
go
begin transaction
drop index authors.au_id_ind
commit
go
use master
go
sp_dboption pubs2, "ddl in tran", false
go
use pubs2
go
checkpoint
go
```

Recovery strategies using recovery fault isolation

There are two major strategies for returning a database with suspect pages to a consistent state while users are accessing it: reload and repair.

Both strategies require:

- A clean database dump
- A series of reliable transaction log dumps up to the point at which the database is recovered with suspect pages
- A transaction log dump to a device immediately after the database is recovered to capture changes to the offline pages
- Continuous transaction log dumps to devices while users work in the partially offline database

Reload strategy

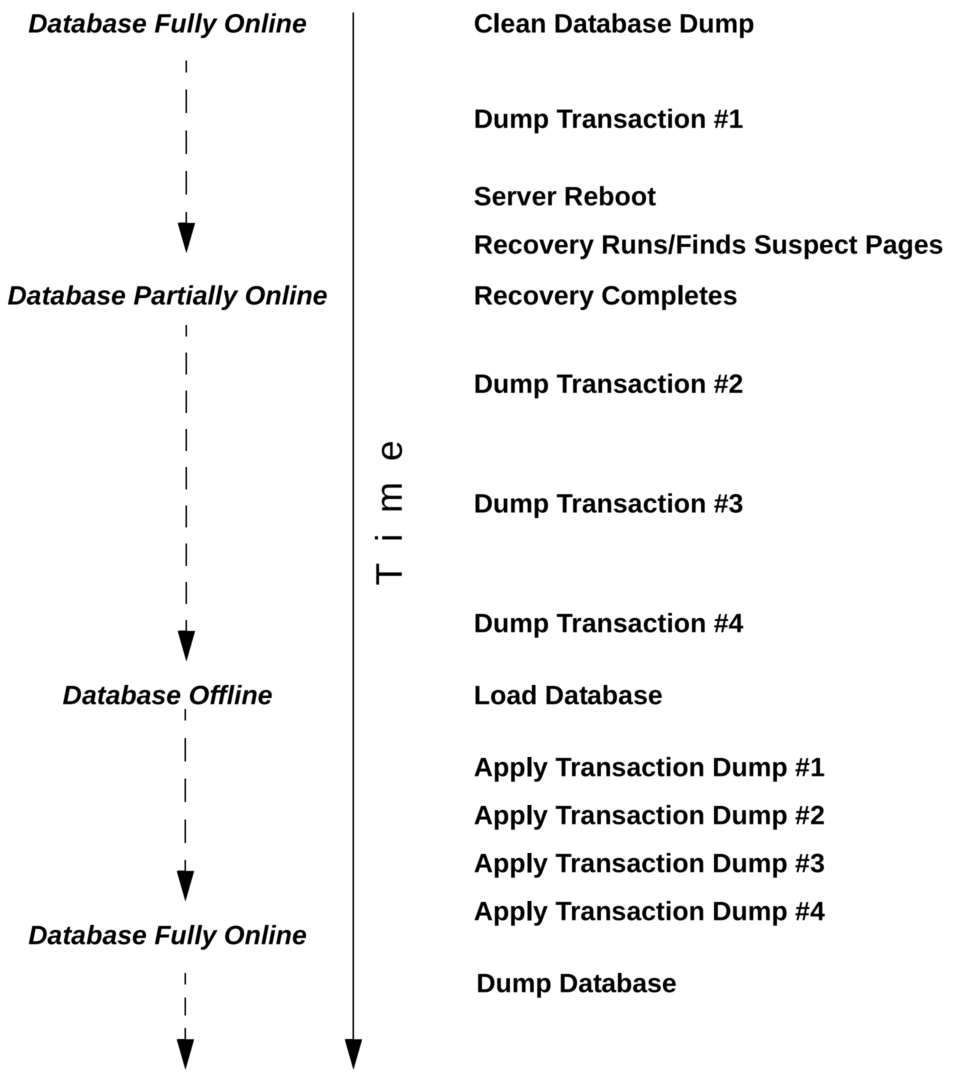
Reloading involves restoring a clean database from backups. When convenient, load the most recent clean database dump, and apply the transaction logs to restore the database.

load database clears the suspect page information from the master.dbo.sysdatabases and master.dbo.sysattributes system tables.

When the restored database is online, dump the database immediately.

Figure 26-1 illustrates the strategy used to reload databases.

Figure 26-1: Reload strategy



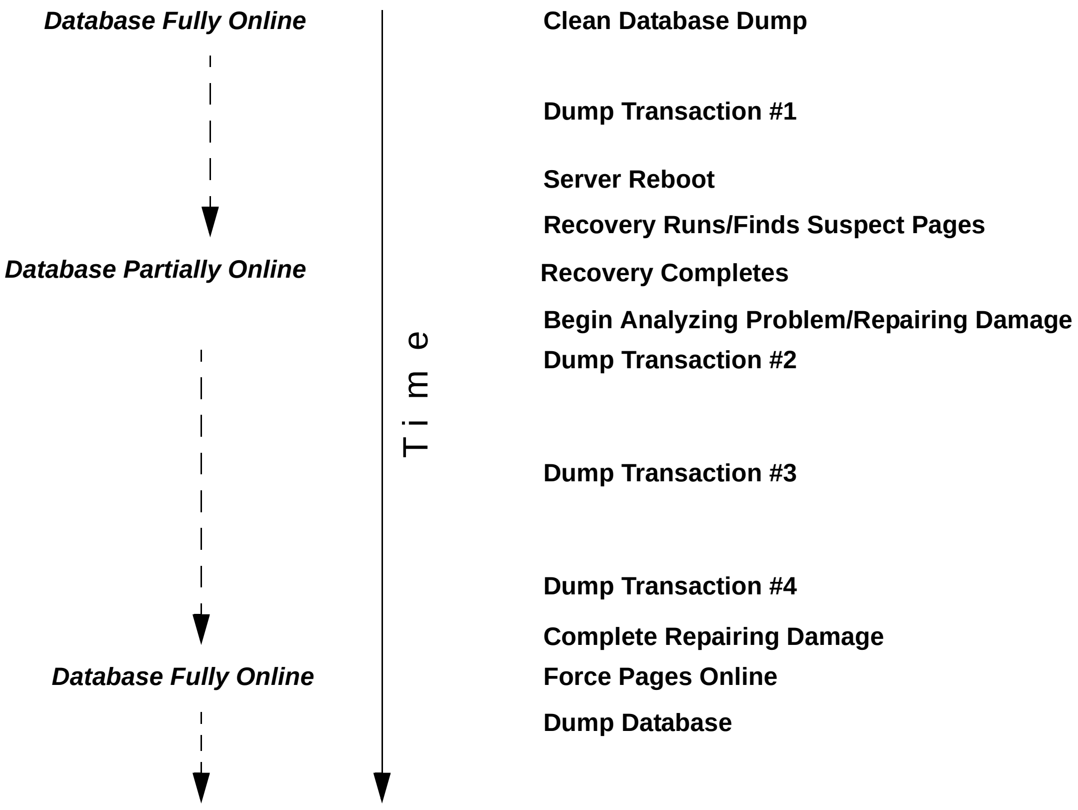
Repair strategy

The repair strategy involves repairing the corrupt pages while the database is partially offline. You diagnose and repair problems using known methods, including dbcc commands, running queries with known results against the suspect pages, and calling Sybase Technical Support, if necessary. Repairing damage can also include dropping and re-creating objects that contain suspect pages.

You can either use sp_forceonline_page to bring offline pages online individually, as they are repaired, or wait until all the offline pages are repaired and bring them online all at once with sp_forceonline_db.

The repair strategy does not require taking the entire database offline. Figure 26-2 illustrates the strategy used to repair corrupt pages.

Figure 26-2: Repair strategy



Assessing the extent of corruption

You can sometimes use recovery fault isolation to assess the extent of corruption by forcing recovery to run and examining the number of pages marked suspect and the objects to which they belong.

For example, if users report problems in a particular database, set the recovery isolation mode to "page," and force recovery by restarting Adaptive Server. When recovery completes, use sp_listsuspect_db or sp_listsuspect_page to determine how many pages are suspect and which database objects are affected.

If the entire database is marked suspect and you receive the message:

```
Reached suspect threshold ’%d’ for database ’%.*s’.
Increase suspect threshold using
sp_setsuspect_threshold.
```

use sp_setsuspect_threshold to raise the suspect escalation threshold and force recovery to run again. Each time you get this message, you can raise the threshold and run recovery until the database comes online. If you do not get this message, the corruption is not isolated to specific pages, in which case this strategy for determining the number of suspect pages will not work.

Using the dump and load commands

In case of media failure, such as a disk crash, you can restore your databases if—and only if—you have regular backups of the databases and their transaction logs. Full recovery depends on the regular use of the dump database and dump transaction commands to back up databases and the load database and load transaction commands to restore them. These commands are described briefly below and more fully in Chapter 27, "Backing Up and Restoring User Databases," and Chapter 28, "Restoring the System Databases."

Warning! Never use operating system copy commands to copy a database device. Loading the copy into Adaptive Server causes massive database corruption.

The dump commands can complete successfully even if your database is corrupt. Before you back up a database, use the dbcc commands to check its consistency. See Chapter 25, “Checking Database Consistency,” for more information.

Warning! If you dump directly to tape, do not store any other types of files (UNIX backups, tar files, and so on) on that tape. Doing so can invalidate the Sybase dump files. However, if you dump to a UNIX file system, the resulting files can be archived to a tape.

Making routine database dumps: *dump database*

The dump database command makes a copy of the entire database, including both the data and the transaction log. dump database does *not* truncate the log.

dump database allows **dynamic dumps**. Users can continue to make changes to the database while the dump takes place. This makes it convenient to back up databases on a regular basis.

dump database executes in three phases. A progress message informs you when each phase completes. When the dump is finished, it reflects all changes that were made during its execution, except for those initiated during phase 3.

Making routine transaction log dumps: *dump transaction*

Use the dump transaction command (or its abbreviation, dump tran) to make routine backups of your transaction log. dump transaction is similar to the incremental backups provided by many operating systems. It copies the transaction log, providing a record of any database changes made since the last database or transaction log dump. After dump transaction has copied the log, it truncates the inactive portion.

dump transaction takes less time and storage space than a full database backup, and it is usually run more often. Users can continue to make changes to the database while the dump is taking place. You can run dump transaction only if the database stores its log on a separate segment.

After a media failure, use the with no_truncate option of dump transaction to back up your transaction log. This provides a record of the transaction log up to the time of the failure.

Copying the log after device failure: *dump tran with no_truncate*

If your data device fails and the database is inaccessible, use the with no_truncate option of dump transaction to get a current copy of the log. This option does not truncate the log. You can use it only if the transaction log is on a separate segment and the master database is accessible.

Restoring the entire database: *load database*

Use the load database command to load the backup created with dump database. You can load the dump into a preexisting database or create a new database with the for load option. When you create a new database, allocate at least as much space as was allocated to the original database.

Warning! You cannot load a dump that was made on a different platform or generated on pre-version 10.0 SQL Server. If the database you are loading includes tables that contain the primary keys for tables in other databases, you must load the dump into a database with the same database name as the one dumped.

The load database command sets the database status to “offline.” This means you do not have to use the no chkpt on recovery, dbo use only, and read only options of sp_dboption before you load a database. However, no one can use a database during the database load and subsequent transaction log loads. To make the database accessible to users, issue the online database command.

After the database is loaded, Adaptive Server may need to:

- “Zero” all unused pages, if the database being loaded into is larger than the dumped database.
- Complete recovery, applying transaction log changes to the data.

Depending on the number of unallocated pages or long transactions, this can take a few seconds or many hours for a very large database. Adaptive Server issues messages that it is "zero-ing" pages or has begun recovery. These messages are normally buffered; to see them, issue:

```
set flushmessage on
```

Applying changes to the database: *load transaction*

After you have loaded the database, use the load transaction command (or its abbreviation, load tran) to load each transaction log dump *in the order in which it was made*. This process reconstructs the database by re-executing the changes recorded in the transaction log. If necessary, you can recover a database by rolling it forward to a particular time in its transaction log, using the until_time option of load transaction.

Users cannot make changes to the database between the load database and load transaction commands, due to the "offline" status set by load database.

You can load only the transaction log dumps that are at the same release level as the associated database.

When the entire sequence of transaction log dumps has been loaded, the database reflects all transactions that had committed at the time of the last transaction log dump.

Making the database available to users: *online database*

When the load sequence completes, change the database status to "online," to make it available to users. A database loaded by load database remains inaccessible until you issue the online database command is issued.

Before you issue this command, be sure you have loaded all required transaction logs.

Moving a database to another Adaptive Server

You can use dump database and load database to move a database from one Adaptive Server to another, as long as both Adaptive Servers run on the same hardware and software platform. However, you must ensure that the device allocations on the target Adaptive Server match those on the original. Otherwise, system and user-defined segments in the new database will not match those in the original database.

To preserve device allocations when loading a database dump into a new Adaptive Server, use the same instructions as for recovering a user database from a failed device. See "Examining the space usage" on page 875 for more information.

Also, follow these general guidelines when moving system databases to different devices:

- Before moving the master database, always unmirror the master device. If you do not, Adaptive Server will try to use the old mirror device file when you start Adaptive Server with the new device.
- When moving the master database, use a new device that is the same size as the original to avoid allocation errors in sysdevices.
- To move the sybsecurity database, place the new database in single-user mode before loading the old data into it.

Upgrading a user database

You can load dumps into the current release of Adaptive Server from any version of Adaptive Server that is at version 10.0 and later. The loaded database is not upgraded until you issue online database.

The steps for upgrading user databases are the same as for system databases:

1. Use load database to load a database dump of a release 10.0 or later Adaptive Server. load database sets the database status to "offline."
2. Use load transaction to load, *in order*, all transaction logs generated after the last database dump. Be sure you have loaded all transaction logs before going to step 3.

3 Use online database to upgrade the database. The online database command upgrades the database because its present state is incompatible with the current release of Adaptive Server. When the upgrade completes, the database status is set to “online,” which makes the database available for public use.

4 Make a dump of the upgraded database. A dump database must occur before a dump transaction command is permitted.

For more information about load database, load transaction, and online database, see the *Adaptive Server Reference Manual*.

Using the special *dump transaction* options

In certain circumstances, the simple model described above does not apply. Table 26-1 describes when to use the special with no_log and with truncate_only options instead of the standard dump transaction command.

Warning! Use the special dump transaction commands *only* as indicated in Table 26-1. In particular, use dump transaction with no_log as a last resort and use it only once after dump transaction with no_truncate fails. The dump transaction with no_log command frees very little space in the transaction log. If you continue to load data after entering dump transaction with no_log, the log may fill completely, causing any further dump transaction commands to fail. Use the alter database command to allocate additional space to the database.

Table 26-1: When to use dump transaction with truncate_only or with no_log

When	Use
The log is on the same segment as the data.	dump transaction with truncate_only to truncate the log dump database to copy the entire database, including the log
You are not concerned with the recovery of recent transactions (for example, in an early development environment).	dump transaction with truncate_only to truncate the log dump database to copy the entire database
Your usual method of dumping the transaction log (either the standard dump transaction command or dump transaction with truncate_only) fails because of insufficient log space.	dump transaction with no_log to truncate the log without recording the event dump database immediately afterward to copy the entire database, including the log

Using the special load options to identify dump files

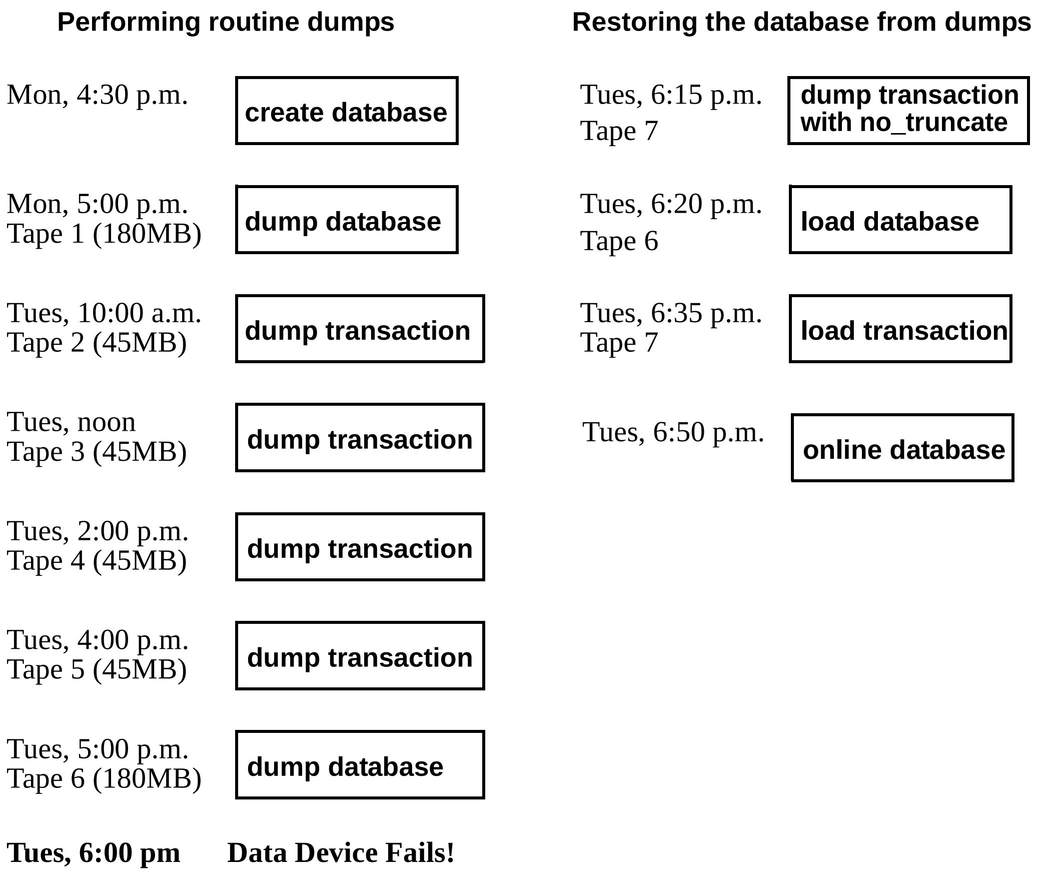
Use the with headeronly option to provide header information for a specified file or for the first file on a tape. Use the with listonly option to return information about all files on a tape. These options do not actually load databases or transaction logs on the tape.

Note These options are mutually exclusive. If you specify both, with listonly prevails.

Restoring a database from backups

Figure 26-3 illustrates the process of restoring a database that is created at 4:30 p.m. on Monday and dumped immediately afterward. Full database dumps are made every night at 5:00 p.m. Transaction log dumps are made at 10:00 a.m., 12:00 p.m., 2:00 p.m., and 4:00 p.m. every day:

Figure 26-3: Restoring a database, a scenario

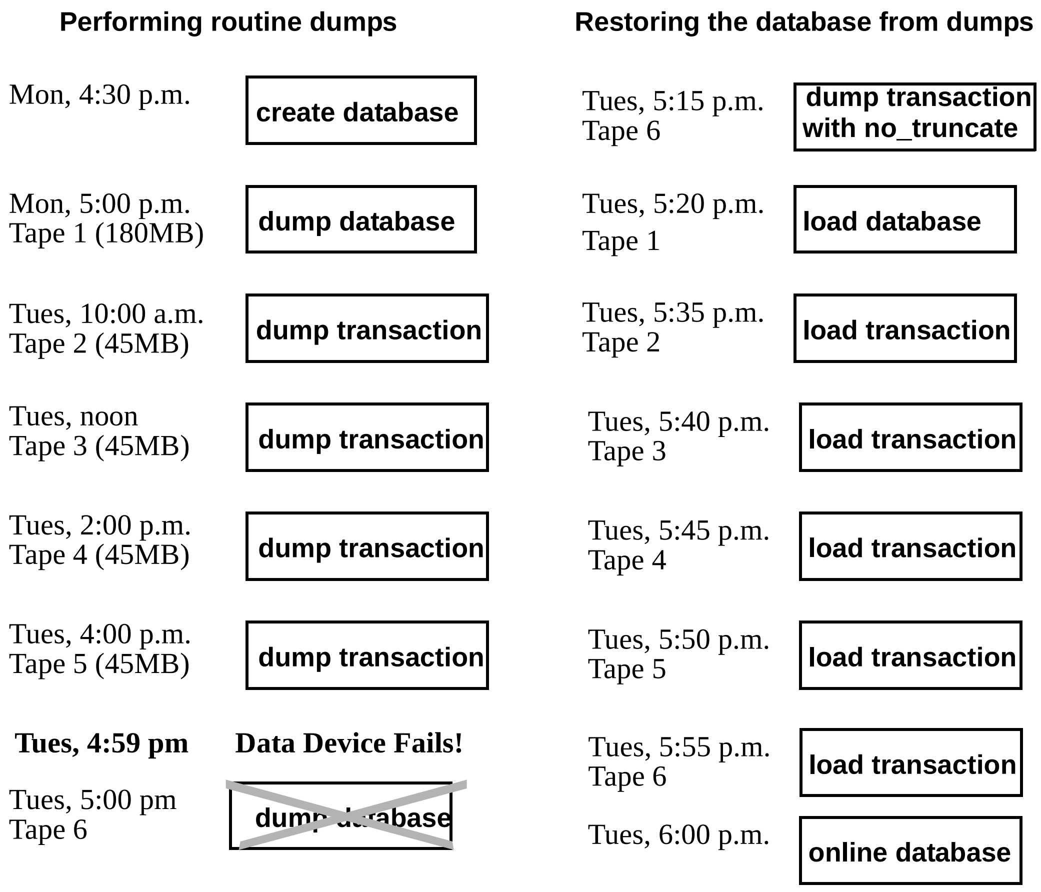


If the disk that stores the data fails on Tuesday at 6:00 p.m., follow these steps to restore the database:

1. Use dump transaction with no_truncate to get a current transaction log dump.
2. Use load database to load the most recent database dump, Tape 6. load database sets the database status to “offline.”
3. Use load transaction to apply the most recent transaction log dump, Tape 7.
4. Use online database to set the database status to “online.”

Figure 26-4 illustrates how to restore the database when the data device fails at 4:59 p.m. on Tuesday—just before the operator is scheduled to make the nightly database dump:

Figure 26-4: Restoring a database, a second scenario



Use the following steps to restore the database:

1. Use dump transaction with no_truncate to get a current transaction log dump on Tape 6 (the tape you would have used for the routine database dump).

2. Use load database to load the most recent database dump, Tape 1. load database sets the database status to "offline."

3 Use load transaction to load Tapes 2, 3, 4, and 5 and the most recent transaction log dump, Tape 6.

4 Use online database to set the database status to “online.”

Suspending and resuming updates to databases

quiesce database hold allows you to block updates to one or more databases while you perform a disk unmirroring or external copy of each database device. Because no writes are performed during this time, the external copy (the secondary image) of the database is identical to the primary image. While the database is in the quiescent state, read-only queries to operations on the database are allowed. To resume updates to the database, issue quiesce database release when the external copy operation has completed. You can load the external copy of the database onto a secondary server, ensuring that you have a transactionally consistent copy of your primary image. You can issue quiesce database hold from one isql connection and then log in with another isql connection and issue quiesce database release.

The syntax for quiesce database is:

```
quiesce database tag_name hold database_name [, database_name]
[for external dump]
```

or,

```
quiesce database tag_name release
```

Where *tag_name* is a user-defined label for the list of databases to hold or release, and *database_name* is the name of a database for which you are suspending updates.

Note *tag_name* must conform to the rules for identifiers. See the *Adaptive Server Reference Manual* for a list of these rules. You must use the same *tag_name* for both quiesce database...hold and quiesce database...release.

For example, to suspend updates to the pubs2 database, enter:

```
quiesce database pubs_tag hold pubs2
```

Adaptive Server writes messages similar to the following to the error log:

```
QUIESCE DATABASE command with tag pubs_tag is being executed by process 9.
```

```
Process 9 successfully executed QUIESCE DATABASE with HOLD option for tag
pubs_tag. Processes trying to issue IO operation on the quiesced database(s)
will be suspended until user executes Quiesce Database command with RELEASE
option.
```

Any updates to the pubs2 database are delayed until the database is released, at which time the updates complete. To release the pubs2 database, enter:

```
quiesce database pubs_tag release
```

After releasing the database, you can bring up the secondary server with the -q parameter if you used the for external dump clause. Recovery makes the databases transactionally consistent, or you can wait to bring the database online and then apply the transaction log.

Guidelines for using quiesce database

The simplest way to use quiesce database is to make a full copy of an entire installation. This ensures that system mappings are consistent. These mappings are carried to the secondary installation when the system databases, that contain them are physically copied as part of quiesce database hold's set of databases. These mappings are fulfilled when all user databases in the source installation are copied as part of the same set. quiesce database allows for eight database names during a single operation. If a source installation has more than eight databases, you can issue multiple instances of quiesce database hold to create multiple concurrent quiescent states for multiple sets of databases.

To create the source installation from scratch, you can use almost identical scripts to create both the primary and secondary installations. The script for the secondary installation might vary in the physical device names passed to the disk init command. This approach requires that updates to system devices on the primary server be reflected by identical changes to the secondary server. For example, if you perform an alter database command on the primary server, you must also perform the same command on the secondary server using identical parameters. This approach requires that the database devices be supported by a volume manager, which can present to both the primary and secondary servers the same physical device names for devices that are physically distinct and separate.

Your site may develop its own procedures for making external copies of database devices. However, Sybase recommends the following:

- You include the master database in quiesce database's list of databases.
- You name devices using identical strings on both the primary and secondary servers.
- You make the environments for the master, model, and sybsystemprocs system databases in the primary and secondary installations identical. In particular, sysusages mappings and database IDs for the copied databases must be identical on the primary and secondary servers, and database IDs for both servers must be reflected identically in sysdatabases.
- The mapping between syslogins.suid and sysusers.suid must remain consistent in the secondary server.
- If the primary server and the secondary server share a copy of master, and if the sysdevices entry for each copied device uses identical strings, the *physname* values in both servers must be physically distinct and separate.
- Making external copies of a database has the following restrictions:
    - The copy process can begin only after quiesce database hold has completed.
    - Every device for every database in quiesce database's list of databases must be copied.
    - The external copy must finish before you invoke quiesce database release.
- During the interval that quiesce database provides for the external copy operation, updates are prevented on any disk space belonging to any database in quiesce database's list of databases. This space is defined in sysusages. However, if space on a device is shared between a database in quiesce database's list of databases and a database not in the list, updates to the shared device may occur while the external copy is made. When you are deciding where to locate databases in a system in which you plan to make external copies, you can either:
    - Segregate databases so they do not share devices in an environment where you will use quiesce database, or

    - Plan to copy all the databases on the device (this follows the recommendation above that you make a copy of the entire installation).

- Use quiesce database only when there is little update activity on the databases (preferably during a moment of read-only activity). When you quiesce the database during a quiet time, not only are fewer users inconvenienced, but, depending on the third-party I/O subsystem that is to perform the external copy, there may also be less time spent synchronizing devices involved in the copy operation.

Maintaining server roles in a primary and secondary relationship

If your site consists of two Adaptive Servers, one functioning as the primary server, and the other acting as a secondary server that receives external copies of the primary server’s databases, you must never mix the roles of these servers. That is, the role each server plays can change (the primary server can become the secondary server and vice-versa), but these roles cannot be fulfilled by the same server simultaneously.

Starting the secondary server with the *-q* option

The dataserver -q option identifies the secondary server. Do not use the -q option to start the primary server. Under the -q option, user databases that were copied during quiesce database for external dump stay offline until:

- You dump the transaction log for a database on the primary server with standby access (that is, dump tran with standby_access) followed by load tran with standby_access to the copy of this database on the secondary server, and then perform online database for standby access on this database.
- You force the database online for read and write access by issuing online database. However, if you do this, the database recovery writes compensation log records, and you cannot load the transaction log without either loading the database, or making a new copy of the primary devices using quiesce database.

System databases come online regardless of the -q option, and write compensation log records for any transactions that are rolled back.

“in quiesce” database log record value updated

If you start the secondary server using the -q option of dataserver, for each user database marked internally as “in quiesce”, Adaptive Server issues a message at start-up stating that the database is `“in quiesce.”`

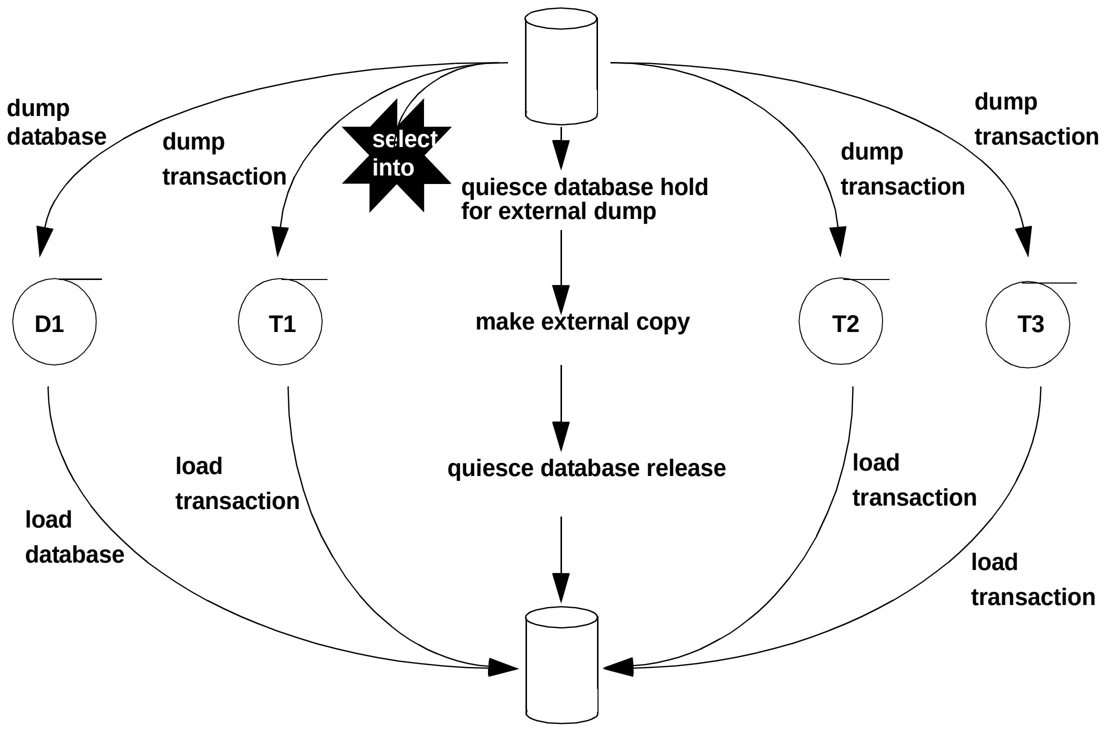
-q recovery for databases copied with quiesce database for external dump acts much like the recovery for load database. Like recovery for load database, it internally records the address of the current last log record, so that a subsequent load transaction can compare this address to the address of the previous current last log record. If these two values do not match, then there has been original activity in the secondary database, and Adaptive Server raises error number 4306.

Updating the dump sequence number

Like dump database, quiesce database updates the dump sequence numbers if there have been non-logged writes. This prevents you from using an earlier database dump or external copy as an improper foundation for a dump sequence.

For example, in the warm standby method that is described in Figure 26-5, archives are produced by dump database (D1), dump transaction (T1), quiesce database, dump transaction (T2), and dump transaction (T3):

Figure 26-5: Warm standby dump sequence



Typically, in an environment with logged updates and no dump tran with truncate_only, you could load D1, T1, T2, and T3 in turn, bypassing any quiesce database hold. This approach is used in a warm standby situation, where succeeding database dumps on the primary server simplify media failure recovery scenarios. On the secondary, or standby server, which is used for decision support systems, you may prefer continuous incremental applications of load transaction instead of interruptions from external copy operation.

However, if an unlogged operation occurs (say, a select into, as happens in Figure 26-5) after the dump transaction that produces T1, a subsequent dump transaction to archive is not allowed, and you must either create another dump of the database, or issue quiesce database for external copy and then make a new external copy of the database. Issuing either of these commands updates the dump sequence number and clears the mark that blocks the dump transaction to archive.

Whether or not you use the for external dump clause depends on how you want recovery to treat the quiescent database that would be marked as `in quiesce`.

quiesce database hold

If you issue quiesce database and do not use the for external dump clause, during the external copy operation that creates the secondary set of databases, the secondary server is not running, and recovery under -q will not see any copied database as "in quiesce." It will recover each server in the normal fashion during start-up recovery; it will not recover them as for load database as was previously described. Subsequently, any attempt to perform a load tran to any of these databases is disallowed with error 4306, `"There was activity on database since last load ..."`, or with error 4305, `"Specified file '%.*s' is out of sequence ..."`

Whether or not there been unlogged activity in the primary database, the dump sequence number will not incremented by quiesce database hold, and the unlogged-writes bits are not cleared by quiesce database release.

quiesce database hold for external dump

When you issue quiesce database for external dump, the external copy of the database "remembers" that it was made during a quiescent interval, so that -q recovery can recover it, as happens for load database. quiesce database release clears this information from the primary database. If non-logged writes have prevented dump tran *to archive* on the primary server, dump tran *to archive* is now enabled.

For any database in quiesce database's list, if non-logged writes have occurred since the previous dump database or quiesce database hold for external dump, the dump sequence number is updated by quiesce database hold for external dump, and the non-logged-writes information is cleared by quiesce database release. The updated sequence number causes load tran to fail if it is applied to a target other than the external copy created under the quiesce database that updated it. This resembles the behavior for dump database of a database with non-logged writes status.

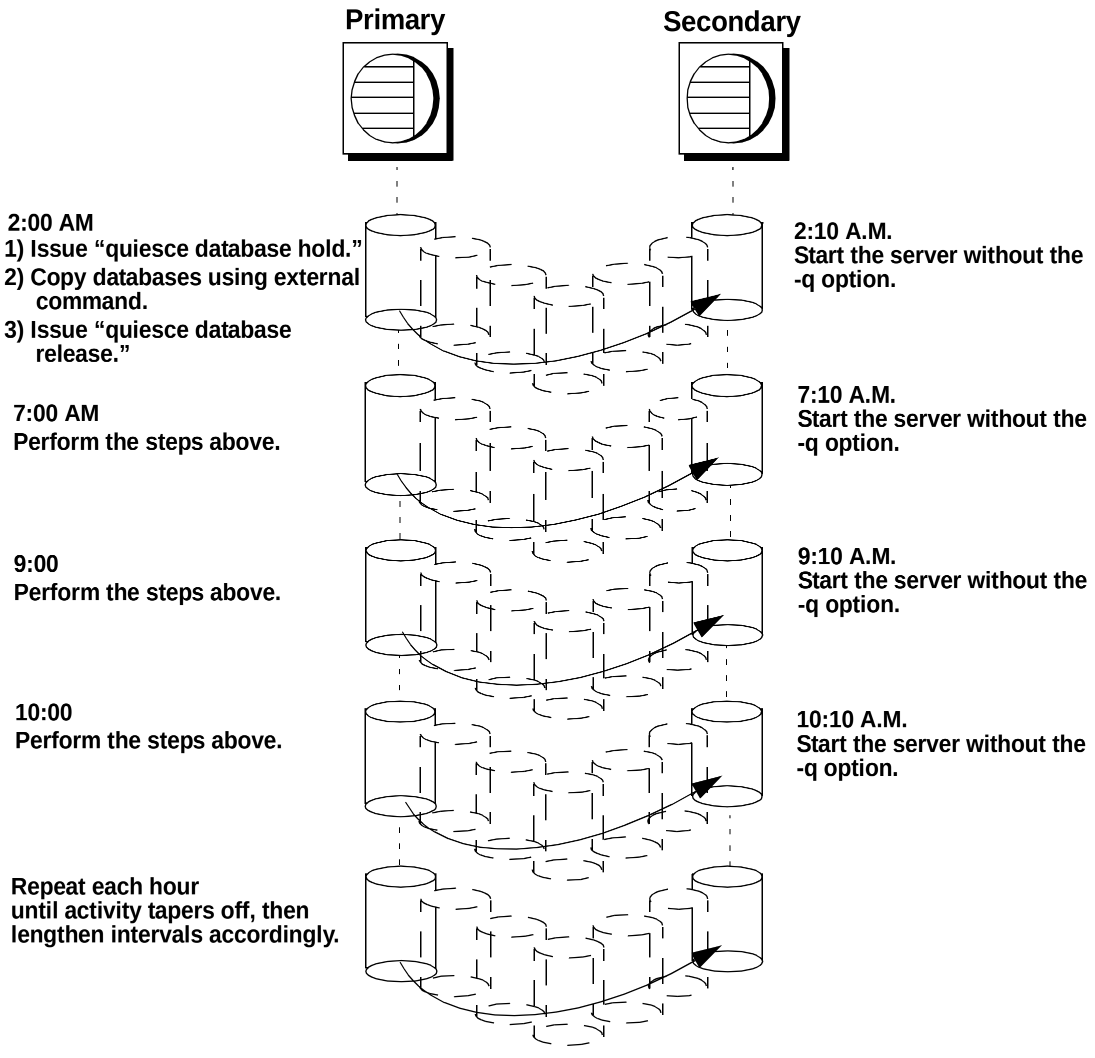
Warning! quiesce database for external dump clears the internal flag that prevents you from performing dump transaction to *archive_device* whether or not you actually make an external copy or perform a database dump. quiesce database has no way of knowing whether or not you have made an external copy. *It is incumbent upon you to perform this duty*. If you use quiesce database hold for external dump to effect a transient write protection rather than to actually perform a copy that serves as the foundation for a new dump sequence, and your application includes occasional unlogged writes, Adaptive Server may allow you to create transaction log dumps that cannot be used. In this situation, dump transaction to *archive_device* initially succeeds, but future load transaction commands may reject these archives because they are out of sequence.

Backing up primary devices with *quiesce database*

Typically, users back up their databases with quiesce database using one of the following methods. Both allow you to off-load decision support applications from the online transaction processor (OLTP) server during normal operation:

- Iterative refresh of the primary device – copy the primary device to the secondary device at refresh intervals. Quiesce the database before each refresh. A system that provides weekly backups using this system is shown in Figure 26-6:

Figure 26-6: Backup schedule for iterative refresh method

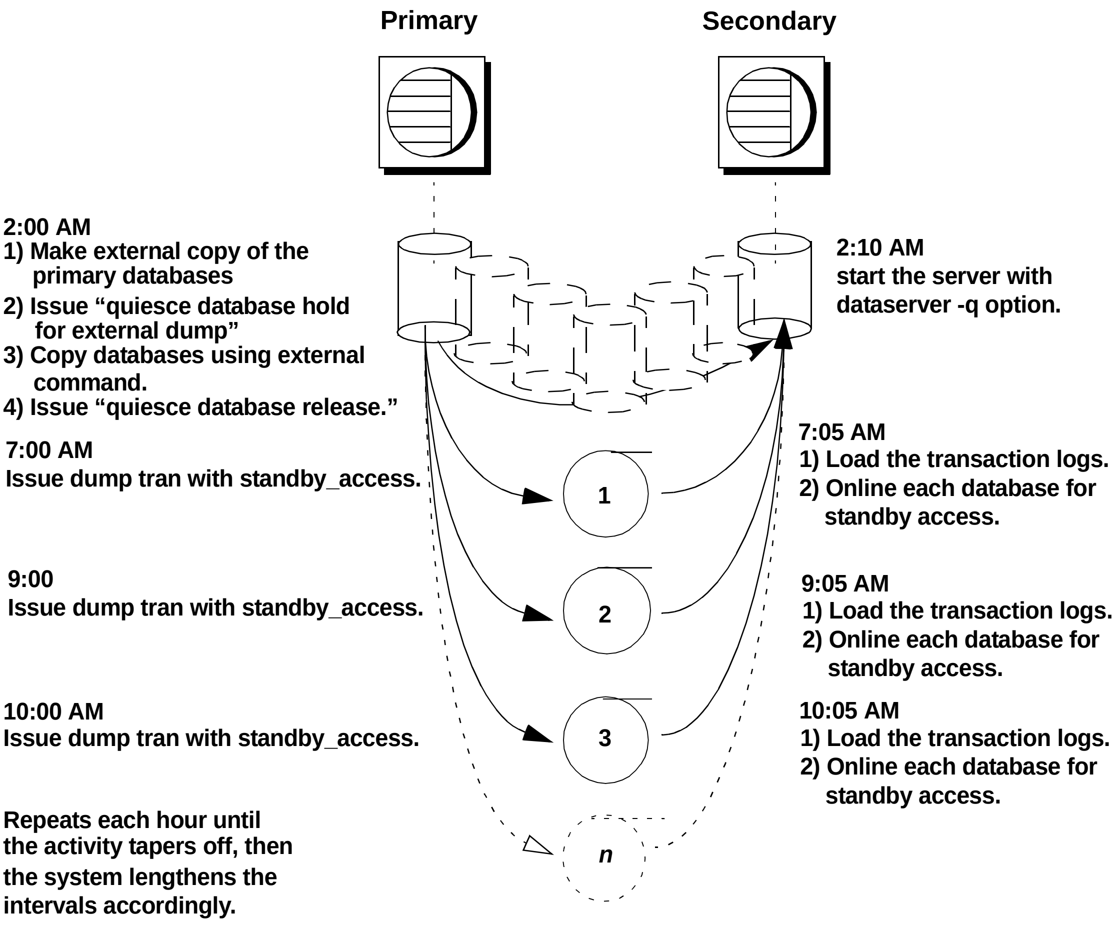

Primary
Secondary
2:00 AM
1) Issue "quiesce database hold."
2) Copy databases using external command.
3) Issue "quiesce database release."
2:10 A.M.
Start the server without the -q option.
7:00 AM
Perform the steps above.
7:10 A.M.
Start the server without the -q option.
9:00
Perform the steps above.
9:10 A.M.
Start the server without the -q option.
10:00
Perform the steps above.
10:10 A.M.
Start the server without the -q option.
Repeat each hour until activity tapers off, then lengthen intervals accordingly.

If you are using the iterative refresh method, you do not have to use the -q option to restart the secondary server (after a crash or system maintenance). Any incomplete transactions generate compensation log records, and the affected databases come online in the regular fashion.

- Warm standby method – allows full concurrency for the OLTP server because it does not block writes.

  After you make an external (secondary) copy of the primary database devices using the for external dump clause, refresh the secondary databases with periodic applications of the transaction logs with dumps from the primary server. For this method, quiesce the databases once to make an external copy of the set of databases and then refresh each periodically using a dump tran with standby_access. A system that uses a daily update of the primary device and then hourly backups of the transaction log is shown in Figure 26-7.

Figure 26-7: Backup schedule for warm standby method


Primary
Secondary
2:00 AM
1) Make external copy of the primary databases
2) Issue "quiesce database hold for external dump"
3) Copy databases using external command.
4) Issue "quiesce database release."
2:10 AM
start the server with dataserver -q option.
7:00 AM
Issue dump tran with standby_access.
1
7:05 AM
1) Load the transaction logs.
2) Online each database for standby access.
9:00
Issue dump tran with standby_access.
2
9:05 AM
1) Load the transaction logs.
2) Online each database for standby access.
10:00 AM
Issue dump tran with standby_access.
3
10:05 AM
1) Load the transaction logs.
2) Online each database for standby access.
Repeats each hour until the activity tapers off, then the system lengthens the intervals accordingly.
n

Recovery of databases for warm standby method

If you are using the warm standby method, Adaptive Server must know whether it is starting the primary or the secondary server. Use the -q option of the dataserver command to specify that you are starting the secondary server. If you do not start the server with the -q option:

- The databases are recovered normally, not as they would be for load database.
- Any uncommitted transactions at the time you issue quiesce database are rolled back.

See "Starting the secondary server with the -q option" on page 797 for more information.

The recovery sequence proceeds differently, depending on whether the database is marked `in quiesce`.

Recovery of databases that are not marked "in quiesce"

Under the -q option, if a database is not marked `in quiesce`, it is recovered as it would be in the primary server. That is, if the database is not currently in a load sequence from previous operations, it is fully recovered and brought online. If there are incomplete transactions, they are rolled back and compensation log records are written during recovery.

Recovery of databases that are marked as "in quiesce"

- User databases – user databases that are marked `in quiesce` recover in the same manner as databases recovering during load database. This enables load tran to detect any activity that has occurred in the primary database since the server was brought down. After you start the secondary server with the -q option, the recovery process encounters the `in quiesce` mark. Adaptive Server issues a message stating that the database is in a load sequence and is being left offline. If you are using the warm standby method, do not bring the database online for its decision support system role until you have loaded the first transaction dump produced by a dump tran with standby_access. Then use online database for standby_access.
- System databases – system databases come fully online immediately. The `in quiesce` mark is erased and ignored.

Making archived copies during the quiescent state

quiesce database hold for external dump signifies your intent to make external copies of your databases during the quiescent state. Because these external copies are made after you issue quiesce database hold, the database is transactionally consistent because you are assured that no writes occurred during the interval between the quiesce database hold and the quiesce database release, and recovery can be run in the same manner as start-up recovery. This process is described in Figure 26-5.

If the environment does not have unlogged updates and does not include a dump tran with truncate_only, you might load D1, T1, T2, and T3 in turn, bypassing any quiesce database...hold commands. However, if an unlogged operation (such as a select into shown in Figure 26-5) occurs after the dump transaction that produces T1, dump transaction to archive is no longer allowed.

Using the quiesce database hold for external dump clause addresses this problem by clearing the status bits that prevent the next dump transaction to archive and changing the sequence number of the external copy to create a foundation for a load sequence. However, if there have been no non-logged writes, the sequence number is not incremented.

With or without the for external dump clause, you can make external copies of the databases. However, to apply subsequent transaction dumps from the primary to the secondary servers, you must include the for external dump clause, as shown:

```
quiesce database tag_name hold db_name [, db_name]
... [for external dump]
```

For example:

```
quiesce database pubs_tag hold pubs2 for external
dump
```

Assuming the database mappings have not changed since the primary instance of the database were initiated, the steps for making an external dump are for a single database include:

1 Issue the quiesce database hold for external dump command:

   ```
   quiesce database pubs_tag hold pubs2 for external
   dump
   ```

2 Make an external copy of the database using the method appropriate for your site.

3 Issue quiesce database *tag_name* release:

```
quiesce database pubs_tag release
```

Warning! Clearing the status bits and updating the sequence number enables you to perform a dump transaction whether or not you actually make an external copy after you issue quiesce database. Adaptive Server has no way of knowing whether or not you have made an external copy during the time between quiesce database... hold for external dump and quiesce database... release. If you use the quiesce database hold for external dump command to effect a transient write protection rather than to actually perform a copy that can serve as the foundation for a new dump sequence, and your application includes occasional unlogged writes, Adaptive Server allows you to create transaction log dumps that cannot be used. dump transaction to *archive_device* succeeds, but load transaction rejects these archives as being out of sequence.

Designating responsibility for backups

Many organizations have an operator who performs all backup and recovery operations. Only a System Administrator, a Database Owner, or an operator can execute the dump and load commands. The Database Owner can dump only his or her own database. The operator and System Administrator can dump and load any database.

Any user can execute sp_volchanged to notify the Backup Server when a tape volume is changed.

Using the Backup Server for backup and recovery

Dumps and loads are performed by an Open Server program, Backup Server, running on the same machine as Adaptive Server. You can perform backups over the network, using a Backup Server on a remote computer and another on the local computer.

Note Backup Server cannot dump to multi-disk volumes.

Backup Server:

- Creates and loads from “striped dumps.” **Dump striping** allows you to use up to 32 backup devices in parallel. This splits the database into approximately equal portions and backs up each portion to a separate device.
- Creates and loads single dumps that span several tapes.
- Dumps and loads over the network to a Backup Server running on another machine.
- Dumps several databases or transaction logs onto a single tape.
- Loads a single file from a tape that contains many database or log dumps.
- Supports platform-specific tape handling options.
- Directs volume-handling requests to the session where the dump or load command was issued or to its operator console.
- Detects the physical characteristics of the dump devices to determine protocols, block sizes, and other characteristics.

Relationship between Adaptive Server and Backup Servers

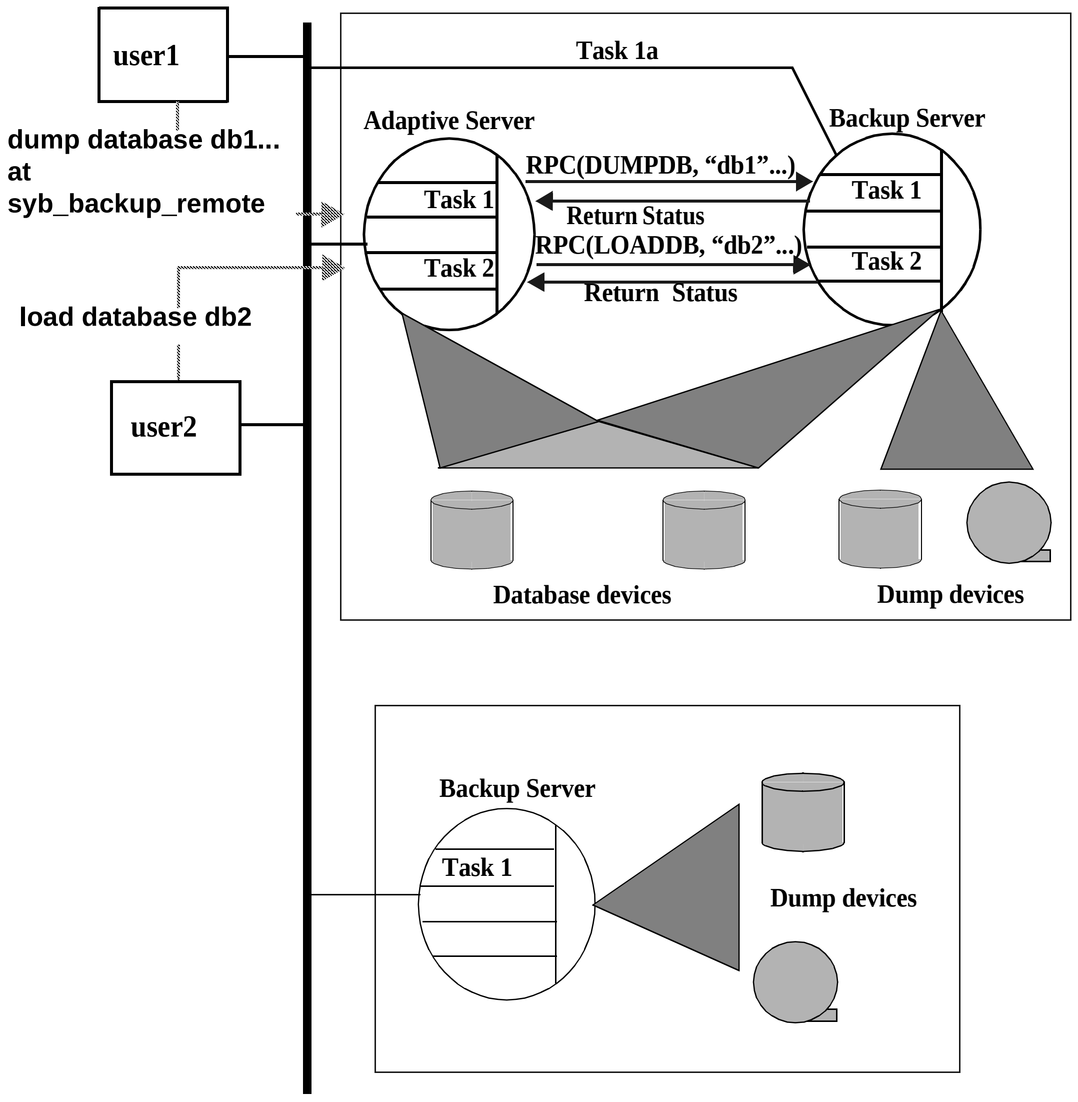
Figure 26-8 shows two users performing backup activities simultaneously on two databases:

- User1 is dumping database `db1` to a remote Backup Server.
- User2 is loading database `db2` from the local Backup Server.

Each user issues the appropriate dump or load command from a Adaptive Server session. Adaptive Server interprets the command and sends remote procedure calls (RPCs) to the Backup Server. The calls indicate which database pages to dump or load, which dump devices to use, and other options.

While the dumps and loads execute, Adaptive Server and Backup Server use RPCs to exchange instructions and status messages. Backup Server—not Adaptive Server—performs all data transfer for the dump and load commands.

Figure 26-8: Adaptive Server and Backup Server with remote Backup Server



When the local Backup Server receives user1's dump instructions, it reads the specified pages from the database devices and sends them to the remote Backup Server. The remote Backup Server saves the data to offline media.

Simultaneously, the local Backup Server performs user2's load command by reading data from local dump devices and writing it to the database device.

Communicating with the Backup Server

To use the dump and load commands, an Adaptive Server must be able to communicate with its Backup Server. These are the requirements:

- The Backup Server must be running on the same machine as the Adaptive Server (or on the same cluster for OpenVMS).
- The Backup Server must be listed in the master..sysservers table. The Backup Server entry, SYB_BACKUP, is created in sysservers when you install Adaptive Server. Use sp_helpserver to see this information.
- The Backup Server must be listed in the interfaces file. The entry for the local Backup Server is created when you install Adaptive Server. The name of the Backup Server listed in the interfaces file must match the column srvnet name for the SYB_BACKUP entry in master..sysservers. If you have installed a remote Backup Server on another machine, create the interfaces file on a file system that is shared by both machines, or copy the entry to your local interfaces file. The name of the remote Backup Server must be the same in both interfaces files.
- The user who starts the Backup Server process must have write permission for the dump devices. The "sybase" user, who usually starts Adaptive Server and Backup Server, can read from and write to the database devices.
- Adaptive Server must be configured for remote access. By default, Adaptive Server is installed with remote access enabled. See "Configuring your server for remote access" on page 811 for more information.

Mounting a new volume

Note Backup Server cannot dump to multi-disk volumes.

During the backup and restore process, change tape volumes. If the Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies the Backup Server by executing sp_volchanged on Adaptive Server.

On UNIX systems, the Backup Server requests a volume change when the tape capacity has been reached. The operator mounts another tape and then executes sp_volchanged (see Table 26-2).

Table 26-2: Changing tape volumes on a UNIX system

Sequence	Operator using *isql*	Adaptive Server	Backup Server
1	Issues the dump database command		
2		Sends dump request to Backup Server	
3			Receives dump request message from Adaptive Server Sends message for tape mounting to operator Waits for operator's reply
4	Receives volume change request from Backup Server Mounts tapes Executes sp_volchanged		
5			Checks tapes If tapes are okay, begins dump When tape is full, sends volume change request to operator
6	Receives volume change request from Backup Server Mounts tapes Executes sp_volchanged		
7			Continues dump When dump is complete, sends messages to operator and Adaptive Server

Sequence	Operator using *isql*	Adaptive Server	Backup Server
8	Receives message that dump is complete Removes and labels tapes	Receives message that dump is complete Releases locks Completes the dump database command	

Starting and stopping Backup Server

Most UNIX systems use the startserver utility to start Backup Server on the same machine as Adaptive Server. On Windows NT, you can start Backup Server from Sybase Central. See the configuration documentation for your platform for information about starting Backup Server.

Use shutdown to shut down a Backup Server. See Chapter 4, “Diagnosing System Problems,” and the *Adaptive Server Reference Manual* for information about this command.

Configuring your server for remote access

The remote access configuration parameter is set to 1 when you install Adaptive Server. This allows Adaptive Server to execute remote procedure calls to the Backup Server.

For security reasons, you may want to disable remote access except when dumps and loads are taking place. To disable remote access, use:

```
sp_configure "allow remote access", 0
```

Before you perform a dump or load, use the following command to re-enable remote access:

```
sp_configure "allow remote access", 1
```

allow remote access is dynamic and does not require a restart of Adaptive Server to take effect. Only a System Security Officer can set allow remote access.

Choosing backup media

Tapes are preferred as dump devices, since they permit a library of database and transaction log dumps to be kept offline. Large databases can span multiple tape volumes. On UNIX systems, the Backup Server requires nonrewinding tape devices for all dumps and loads.

For a list of supported dump devices, see the the configuration documentation for your platform.

Protecting backup tapes from being overwritten

The tape retention in days configuration parameter determines how many days' backup tapes are protected from being overwritten. The default value of tape retention in days is 0. Which means that backup tapes can be overwritten immediately.

Use sp_configure to change the tape retention in days value. The new value takes effect the next time you restart Adaptive Server:

```
sp_configure "tape retention in days", 14
```

Both dump database and dump transaction provide a retaindays option that overrides the tape retention in days value for that dump.

Dumping to files or disks

In general, dumping to a file or disk is not recommended. If the disk or computer containing that file crashes, there may be no way to recover the dumps. On UNIX and PC systems, the entire master database dump must fit into a single volume. On these systems, dumping to a file or disk is your only option if the master database is too large to fit on a single tape volume, unless you have a second Adaptive Server that can issue sp_volchanged requests.

Dumps to a file or disk can be copied to tape for offline storage, but these tapes must be copied back to an online file before they can be read by Adaptive Server. Backup Server cannot directly read a dump that is made to a disk file and then copied to tape.

Creating logical device names for local dump devices

If you are dumping to or loading from local devices (that is, if you are not performing backups over a network to a remote Backup Server), you can specify dump devices either by providing their physical locations or by specifying their logical device names. In the latter case, you may want to create logical dump device names in the sysdevices system table of the master database.

Note If you are dumping to or loading from a remote Backup Server, you must specify the absolute path name of the dump device. You cannot use a logical device name.

The sysdevices table stores information about each database and backup device, including its *physical_name* (the actual operating system device or file name) and its *device_name* (or logical name, known only within Adaptive Server). On most platforms, Adaptive Server has one or two aliases for tape devices installed in sysdevices. The physical names for these devices are common disk drive names for the platform; the logical names are tapedump1 and tapedump2.

When you create backup scripts and threshold procedures, use logical names, rather than physical device names, and whenever possible, you must modify scripts and procedures that refer to actual device names each time you replace a backup device. If you use logical device names, you can simply drop the sysdevices entry for the failed device and create a new entry that associates the logical name with a different physical device.

Listing the current device names

To list the backup devices for your system, run:

```
select * from master..sysdevices
  where status = 16 or status = 24
```

To list both the physical and logical names for database and backup devices, use sp_helpdevice:

```
sp_helpdevice tapedump1
device_name physical_name
 description
 status cntrltype device_number low      high
 ------ --------- ------------- -------- -------
```

```
tapedump1 /dev/nrmt4
tape, 625 MB, dump device
   16          3             0         0   20000
```

Adding a backup device

Use sp_addumpdevice to add a backup device:

> sp_addumpdevice{ "tape" | "disk"} , *logicalname*, *physicalname*,
> *tapesize*

physicalname can be either an absolute path name or a relative path name. During dumps and loads, the Backup Server resolves relative path names by looking in Adaptive Server's current working directory.

tapesize is the capacity of the tape in megabytes. Most platforms require this parameter for tape devices but ignore it for disk devices. The Backup Server uses the *tapesize* parameter if the dump command does not specify a tape capacity.

tapesize must be at least 1MB and should be slightly below the capacity rated for the device.

Redefining a logical device name

To use an existing logical device name for a different physical device, drop the device with sp_dropdevice and then add it with sp_addumpdevice. For example:

```
sp_dropdevice tapedump2
sp_addumpdevice "tape", tapedump2, "/dev/nrmt8", 625
```

Scheduling backups of user databases

A major task in developing a backup plan is determining how often to back up your databases. The frequency of your backups determines how much work you will lose in the event of a media failure. This section presents some guidelines about when to dump user databases and transaction logs.

Scheduling routine backups

Dump each user database just after you create it, to provide a base point, and on a fixed schedule thereafter. Daily backups of the transaction log and weekly backups of the database are the minimum recommended. Many installations with large and active databases make database dumps every day and transaction log dumps every half hour or hour.

Interdependent databases—databases where there are cross-database transactions, triggers, or referential integrity—should be backed up at the same time, during a period when there is no cross-database data modification activity. If one of these databases fails and needs to be reloaded, they should all be reloaded from these simultaneous dumps.

Warning! Always dump both databases immediately after adding, changing, or removing a cross-database constraint or dropping a table that contains a cross-database constraint.

Other times to back up a database

In addition to routine dumps, you should dump a database each time you upgrade a user database, create a new index, perform an unlogged operation, or run the dump transaction with no_log or dump transaction with truncate_only command.

Dumping a user database after upgrading

After you upgrade a user database to the current version of Adaptive Server, dump the newly upgraded database to create a dump that is compatible with the current release. A dump database must occur on upgraded user databases before a dump transaction is permitted.

Dumping a database after creating an index

When you add an index to a table, create index is recorded in the transaction log. As it fills the index pages with information, however, Adaptive Server does not log the changes.

If your database device fails after you create an index, load transaction may take as long to reconstruct the index as create index took to build it. To avoid lengthy delays, dump each database immediately after creating an index on one of its tables.

Dumping a database after unlogged operations

Adaptive Server writes the data for the following commands directly to disk, adding no entries (or, in the case of bcp, minimal entries) in the transaction log:

- Non-logged writetext
- select into on a permanent table
- Fast bulk copy (bcp) into a table with no triggers or indexes

You cannot recover any changes made to the database after issuing one of these commands. To ensure that these commands are recoverable, issue a dump database command immediately after executing any of these commands.

Dumping a database when the log has been truncated

dump transaction with truncate_only and dump transaction with no_log remove transactions from the log without making a backup copy. To ensure recoverability, you must dump the database each time you run either command because of lack of disk space. You cannot copying the transaction log until you have done so. See “Using the special dump transaction options” on page 790 for more information.

If the trunc log on chkpt database option is set to true, and the transaction log contains 50 rows or more, Adaptive Server truncates the log when an automatic checkpoint occurs. If this happens, you must dump the entire database—not the transaction log—to ensure recoverability.

Scheduling backups of *master*

Back up the master database *regularly and frequently.*

Backups of the master database are used as part of the recovery procedure in case of a failure that affects the master database. If you do not have a current backup of master, you may have to reconstruct vital system tables at a time when you are under pressure to get your databases up and running again.

Dumping *master* after each change

Although you can restrict the creation of database objects in master, system procedures such as sp_addlogin and sp_droplogin, sp_password, and sp_modifylogin allow users to modify system tables in the database. Back up the master database frequently to record these changes.

Back up the master database after each command that affects disks, storage, databases, or segments. Always back up master after issuing any of the following commands or system procedures:

- disk init, sp_addumpdevice, or sp_dropdevice
- Disk mirroring commands
- The segment system procedures sp_addsegment, sp_dropsegment, or sp_extendsegment
- create procedure or drop procedure
- sp_logdevice
- sp_configure
- create database or alter database

Saving scripts and system tables

For further protection, save the scripts containing all of your disk init, create database, and alter database commands and make a hard copy of your sysdatabases, sysusages, and sysdevices tables each time you issue one of these commands.

You cannot use the dataserver command to automatically recover changes that result from these commands. If you keep your scripts—files containing Transact-SQL statements—you can run them to re-create the changes. Otherwise, you must reissue each command against the rebuilt master database.

You should also keep a hard copy of syslogins. When you recover master from a dump, compare the hard copy to your current version of the table to be sure that users retain the same user IDs.

For information on the exact queries to run against the system tables, see “Backing up master and keeping copies of system tables” on page 24.

Truncating the *master* database transaction log

Since the master database transaction log is on the same database devices as the data, you cannot back up its transaction log separately. You cannot move the log of the master database. You must always use dump database to back up the master database. Use dump transaction with the truncate_only option periodically (for instance, after each database dump) to purge the transaction log of the master database.

Avoiding volume changes and recovery

When you dump the master database, be sure that the entire dump fits on a single volume, unless you have more than one Adaptive Server that can communicate with your Backup Server. You must start Adaptive Server in single-user mode before loading the master database. This does not allow a separate user connection to respond to Backup Server’s volume change messages during the load. Since master is usually small in size, placing its backup on a single tape volume is typically not a problem.

Scheduling backups of the *model* database

Keep a current database dump of the model database. Each time you make a change to the model database, make a new backup. If model is damaged and you do not have a backup, you must reenter all the changes you have made to restore model.

Truncating the *model* database’s transaction log

model, like master, stores its transaction log on the same database devices as the data. You must always use dump database to back up the model database and dump transaction with truncate_only to purge the transaction log after each database dump.

Scheduling backups of the *sybsystemprocs* database

The sybsystemprocs database stores only system procedures. Restore this database by running the installmaster script, unless you make changes to the database.

If you change permissions on some system procedures, or create your own system procedures in sybsystemprocs, your two recovery choices are:

- Run installmaster, then reenter all of your changes by re-creating your procedures or by re-executing the grant and revoke commands.
- Back up sybsystemprocs each time you make a change to it.

Both of these recovery options are described in Chapter 28, “Restoring the System Databases.”

Like other system databases, sybsystemprocs stores its transaction log on the same device as the data. You must always use dump database to back up sybsystemprocs. By default, the trunc log on chkpt option is set to true (on) in sybsystemprocs, so you should not need to truncate the transaction log. If you change this database option, be sure to truncate the log when you dump the database.

If you are running on a UNIX system or PC, and you have only one Adaptive Server that can communicate with your Backup Server, be sure that the entire dump of sybsystemprocs fits on a single dump device. Signaling volume changes requires sp_volchanged, and you cannot use this procedure on a server while sybsystemprocs is in the process of recovery.

Configuring Adaptive Server for simultaneous loads

Adaptive Server can perform multiple load and dump commands simultaneously. Loading a database requires one 16K buffer for each active database load. By default, Adaptive Server is configured for six simultaneous loads. To perform more loads simultaneously, a System Administrator can increase the value of number of large i/o buffers:

```
sp_configure "number of large i/o buffers", 12
```

This parameter requires a restart of Adaptive Server. See "number of large i/o buffers" on page 95 for more information. These buffers are not used for dump commands or for load transaction.

Gathering backup statistics

Use dump database to make several practice backups of an actual user database and dump transaction to back up a transaction log. Recover the database with load database and apply successive transaction log dumps with load transaction.

Keep statistics on how long each dump and load takes and how much space it requires. The more closely you approximate real-life backup conditions, the more meaningful your predictions will be.

After you have developed and tested your backup procedures, commit them to paper. Determine a reasonable backup schedule and adhere to it. If you develop, document, and test your backup procedures ahead of time, you will be much better prepared to get your databases online if disaster strikes.

CHAPTER 27

Backing Up and Restoring User Databases

Regular and frequent backups are your only protection against database damage that results from failure of your database devices.

This chapter includes these topics:

Dump and load command syntax

The dump database, dump transaction, load database, and load transaction commands have parallel syntax. Routine dumps and loads require the name of a database and at least one dump device. The commands can also include the following options:

- compress::*compression_level*:: to compress your dump files
- at *server_name* to specify the remote Backup Server
- density, blocksize, and capacity to specify tape storage characteristics
- dumpvolume to specify the volume name of the ANSI tape label
- file = *file_name* to specify the name of the file to dump to or load from
- stripe on *stripe_device* to specify additional dump devices
- dismount, unload, init, and retaindays to specify tape handling
- notify to specify whether Backup Server messages are sent to the client that initiated the dump or load or to the operator_console

Table 27-1 shows the syntax for routine database and log dumps and for dumping the log after a device failure. It indicates what type of information is provided by each part of the dump database or dump transaction statement.

Table 27-1: Syntax for routine dumps and log dumps after device failure

Information provided	**Task**	
	Routine database or log dump	**Log dump after device failure**
Command	`dump {database | transaction}`	`dump transaction`
Database name	*database_name*	*database_name*

Compression	`to`
	`[compress::[`*compression_level*`::]]`
		`to`
		`[compress::[`*compression_level*`::]]`

Dump device	*stripe_device*	*stripe_device*
Remote Backup Server	`[at` *server_name*`]`	`[at` *server_name*`]`

Tape device characteristics	`[density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`]`
		`[density =` *density*`,`
		`blocksize =` *number_bytes*`,`
		`capacity =` *number_kilobytes*`]`

Volume name	`[, dumpvolume =` *volume_name*`]`	`[, dumpvolume =` *volume_name*`]`
File name	`[, file =` *file_name*`]`	`[, file =` *file_name*`]`

Information provided	**Task**	
	Routine database or log dump	**Log dump after device failure**
Characteristics of additional devices (up to 31 devices; one set per device)	[stripe on [compress::[*compression_level*::]] stripe_device [at *server_name*] [density = *density*, blocksize = *number_bytes*, capacity = *number_kilobytes,* file = *file_name*, dumpvolume = *volume_name*]]...	[stripe on [compress::[*compression_level*::]] stripe_device [at *server_name*] [density = *density*, capacity = *number_kilobytes,* file = *file_name*, dumpvolume = *volume_name*]]...

Options that apply to entire dump	[with {
	density = *density,*
	blocksize = *number_bytes*,
	capacity = *number_kilobytes,*
	file = *file_name*,
	[nodismount | dismount],
	[nounload | unload],
	[retaindays = *number_days*],
	[noinit | init],
	file = *file_name*,
	dumpvolume = *volume_name*
	standby_access
		[with {
		density = *density,*
		blocksize = *number_bytes*,
		capacity = *number_kilobytes,*
		file = *file_name*,
		[nodismount | dismount],
		[nounload | unload],
		[retaindays = *number_days*],
		[noinit | init],
		file = *file_name*,
		dumpvolume = *volume_name*,
		standby_access

Do not truncate log		no_truncate
Message destination	[, notify = {client | operator_console}]}]	[, notify = {client | operator_console}]}]

Table 27-2 shows the syntax for loading a database, applying transactions from the log, and returning information about dump headers and files.

Table 27-2: Syntax for load commands

Information Provided	**Task**	
	Load database or apply recent transactions	**Return header or file information but do not load backup**
Command	load {database | transaction}	load {database | transaction}
Database name	*database_name*	*database_name*
Compression	from [compress::]	from [compress::]
Dump device	*stripe_device*	*stripe_device*
Remote Backup Server	[at *server_name*]	[at *server_name*]
Tape device characteristics	[density = *density*,	[density = *density*,
Volume name	[, dumpvolume = *volume_name*]	[, dumpvolume = *volume_name*]
File name	[, file = *file_name*]	[, file = *file_name*]

Information Provided	**Task**	
	Load database or apply recent transactions	**Return header or file information but do not load backup**
Characteristics of additional devices (up to 31 devices; one set per device)	`[stripe on` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*, `file =` *file_name*, `dumpvolume =` *volume_name*`]]...`	`[stripe on` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*, `file =` *file_name*, `dumpvolume =` *volume_name*`]]...`
Tape handling	`[with{` `[density =` *density*, `dumpvolume =` *volume_name*, `file =` *file_name*, `[nodismount | dismount],` `[nounload | unload]`	`[with{` `[density =` *density*, `dumpvolume =` *volume_name*, `file =` *file_name*, `[nodismount | dismount],` `[nounload | unload]`
Provide header information		`[, headeronly]`
List dump files		`[, listonly [= full]]`
Message destination	`[, notify = {client | operator_console}]}]`	`[, notify = {client | operator_console}]}]`
Do not load open transactions	*standby_access*	

Table 27-3 shows the syntax for truncating a log:

- That is not on a separate segment
- Without making a backup copy
- With insufficient free space to successfully complete a dump transaction or dump transaction with truncate_only command

Table 27-3: Special dump transaction options

Information Provided	**Task**		
	Truncate log on same segment as data	**Truncate log without making a copy**	**Truncate log with insufficient free space**
Command	`dump transaction`	`dump transaction`	`dump transaction`
Database name	*database_name*	*database_name*	*database_name*
Do not copy log	`with truncate_only`	`with truncate_only`	`with no_log`

The remainder of this chapter provides greater detail about the information specified in dump and load commands and volume change messages. Routine dumps and loads are described first, followed by log dumps after device failure and the special syntax for truncating logs without making a backup copy.

For information about the permissions required to execute the dump and load commands, refer to "Designating responsibility for backups" on page 806.

Specifying the database and dump device

At a minimum, all dump and load commands must include the name of the database being dumped or loaded. Commands that dump or load data (rather than just truncating a transaction log) must also include a dump device.

Table 27-4 shows the syntax for backing up and loading a database or log.

Table 27-4: Indicating the database name and dump device

	Backing up a database or log	**Loading a database or log**
Database name Dump device	`dump {database | tran}` *database_name*	`load {database | tran}` *database_name*
	`to [compress::[`*compression_level*`::]]`	`from [compress::]`
Dump device	*stripe_device*	*stripe_device*

	Backing up a database or log	Loading a database or log
	[at *server_name*]	[at *server_name*]
	[density = *density,*	[density = *density*,
	blocksize = *number_bytes,*	dumpvolume = *volume_name*
	capacity = *number_kilobytes,*	file = *file_name*]
	dumpvolume = *volume_name*,	[stripe on
	file = *file_name*]	[compress::]*stripe_device*
	[stripe on	[at *server_name*]
	[compress::[*compression_level*::]]	[density = *density*,
	stripe_device	dumpvolume = *volume_name*,
	[at *server_name*]	file = *file_name*] ...]
	[density = *density*,	[with{
	blocksize = *number_bytes,*	density = *density*,
	capacity = *number_kilobytes,*	dumpvolume = *volume_name*,
	dumpvolume = *volume_name,*	file = *file_name*,
	file = *file_name*] ...]	[nodismount | dismount],
	[with{	[nounload | unload],
	density = *density*,	[notify = {client |
	blocksize = *number_bytes,*	operator_console}]}]
	capacity = *number_kilobytes,*	
	dumpvolume = *volume_name,*	
	file = *file_name*,	
	[nodismount | dismount],	
	[nounload | unload],	
	retaindays = *number_days,*	
	[noinit | init],	
	[notify = {client |	
	operator_console}]	
	standby_access}]	

Rules for specifying database names

You can specify the database name as a literal, a local variable, or a parameter to a stored procedure.

If you are loading a database from a dump:

- The database must exist. You can create a database with the for load option of create database, or load it over an existing database. Loading a database always overwrites all the information in the existing database.

- You do not need to use the same database name as the name of the database you dumped. For example, you can dump the pubs2 database, create another database called pubs2_archive, and load the dump into the new database.

Warning! You should never change the name of a database that contains primary keys for references from other databases. If you must load a dump from such a database and provide a different name, first drop the references to it from other databases.

Rules for specifying dump devices

When you specify a dump device:

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You cannot dump to or load from the "null device" (on UNIX, */dev/null*; on OpenVMS, any device name beginning with NL; not applicable to PC platforms).
- When dumping to or loading from a local device, you can use any of the following forms to specify the dump device:
  - An absolute path name
  - A relative path name
  - A logical device name from the sysdevices system table

  The Backup Server resolves relative path names using Adaptive Server's current working directory.
- When dumping or loading over the network:
  - You must specify the absolute path name of the dump device. You cannot use a relative path name or a logical device name from the sysdevices system table.
  - The path name must be valid on the machine on which the Backup Server is running.
  - If the name includes any characters except letters, numbers, or the underscore (_), you must enclose it in quotes.

- If you dump a transaction log using with standby_access, you must load the dump using with standby_access.

Examples

The following examples use a single tape device for dumps and loads. (It is not necessary to use the same device for dumps and loads.)

On UNIX:

```
dump database pubs2 to "/dev/nrmt4"
load database pubs2 from "/dev/nrmt4"
```

On Windows NT:

```
dump database pubs2 to "\\.\tape0"
load database pubs2 from "\\.\tape0"
```

You can also dump to an operating system file. The following example is for Windows NT:

```
dump database pubs2 to "d:\backups\backup1.dat"
load database pubs2 from "d:\backupbackup1.dat"
```

Tape device determination by backup server

When you issue a dump database or dump transaction command, Backup Server checks whether the device type of the specified dump device is known (supplied and supported internally) by Adaptive Server. If the device is not a known type, Backup Server checks the tape configuration file (default location is *$SYBASE/backup_tape.cfg)* for the device configuration.

If the configuration is found, the dump command proceeds.

If the configuration is not found in the tape device configuration file, the dump command fails with the following error message:

```
Device not found in configuration file. INIT needs
to be specified to configure the device.
```

To configure the device, issue the dump database or dump transaction with the init parameter. Using operating system calls, Backup Server attempts to determine the device's characteristics; if successful, it stores the device characteristics in the tape configuration file.

If Backup Server cannot determine the dump device characteristics, it defaults to one dump per tape. The device cannot be used if the configuration fails to write at least one dump file.

Tape configuration by Backup Server applies only to UNIX platforms.

Tape sevice configuration file

Format

The tape device configuration file contains tape device information that is used only by the dump command.

The format of the file is one tape device entry per line. Fields are separated by blanks or tabs.

Creation

This file is created only when Backup Server is ready to write to it (dump database or dump transaction with init). When Backup Server tries to write to this file for the first time, the following warning message is issued:

```
Warning, unable to open device configuration file
for reading. Operating system error. No such file or
directory.
```

Ignore this message. Backup Server gives this warning and then creates the file and writes the configuration information to it.

Manual editing

The only user interaction with the file occurs when the user receives the following error message:

```
Device does not match the current configuration.
Please reconfigure this tape device by removing the
configuration file entry and issuing a dump with the
INIT qualifier.
```

This means that the tape hardware configuration changed for a device name. Delete the line entry for that device name and issue a dump command, as instructed.

Default location

The default path name for the configuration file is *$SYBASE/backup_tape.cfg*. You can change the default location with the Sybase installation utilities. See the installation documentation for your platform for more information.

Specifying the compress option

The dump command includes a compress option that allows you to compress databases and transaction logs using Backup Server.

Table 27-5 The shows the syntax for dump database … compress and dump transaction … compress commands is:

Table 27-5: Indicating the database name and dump device

	Backing up a database or log	**Loading a database or log**
	`dump {database | tran}` *database_name* `to`	`load {database | tran}` *database_name* `from`
Compress option	`[compress::[`*compression_level*`::]]`	`[compress::]`
	stripe_device `[at` *server_name*`]` `[density =` *density,* `blocksize =` *number_bytes,* `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name*`,` `file =` *file_name*`]` `[stripe on`	*stripe_device*`}` `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name* `file =` *file_name*`]` `[stripe on`
Compress option	`[compress::[`*compression_level*`::]]`	`[compress::]`
	stripe_device `[at` *server_name*`]` `[density =` *density*`,` `blocksize =` *number_bytes,* `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name,* `file =` *file_name*`] ...]` `[with{` `density =` *density*`,` `blocksize =` *number_bytes,* `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name,* `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `retaindays =` *number_days,* `[noinit | init],` `[notify = {client |` `operator_console}]` `standby_access}]`	*stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]` `[with{` `density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `[notify = {client |` `operator_console}]}]`

Syntax

The partial syntax specific to dump database … compress and dump transaction … compress is:

> dump database *database_name*
> to compress::[*compression_level*::]*stripe_device*
> …[stripe on compress::[*compression_level*::]*stripe_device*] …

dump transaction *database_name*
    to compress::[*compression_level*::]*stripe_device*
    …[stripe on compress::[*compression_level*::]*stripe_device*]…

Where *database_name* is the database you are loading into, and compress::*compression_level* is a number between 0 and 9, with 0 indicating no compression, and 9 providing the highest level of compression. If you do not specify *compression_level*, the default is 6. *stripe_device* is the full path to the archive file of the database or transaction log you are compressing. If you do not include a full path for your dump file, Adaptive Server creates a dump file in the directory in which you started Adaptive Server.

Use the stripe on clause to use multiple dump devices for a single dump. See "Specifying additional dump devices: the stripe on clause" on page 851 for more information about the stripe on clause.

Note The compress option works only with local archives; you cannot use the servername option.

Example

```
dump database pubs2 to
    "compress::4::/opt/bin/Sybase/dumps/dmp090100.dmp”

Backup Server session id is:  9.  Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume change
request from the Backup Server.
Backup Server: 4.132.1.1: Attempting to open byte stream device:
'compress::4::/opt/bin/Sybase/dumps/dmp090100.dmp::00'
Backup Server: 6.28.1.1: Dumpfile name 'pubs2002580BD27  ' section number 1
mounted on byte stream
'compress::4::/opt/bin/Sybase/dumps/dmp090100.dmp::00'
Backup Server: 4.58.1.1: Database pubs2: 394 kilobytes DUMPed.
Backup Server: 4.58.1.1: Database pubs2: 614 kilobytes DUMPed.
Backup Server: 3.43.1.1: Dump phase number 1 completed.
Backup Server: 3.43.1.1: Dump phase number 2 completed.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.58.1.1: Database pubs2: 622 kilobytes DUMPed.
Backup Server: 3.42.1.1: DUMP is complete (database pubs2).
```

The *compression_level* must be a number between 0 and 9. The compress option does not recognize numbers outside this range, and treats them as part of the file name while it compresses your files using the default compression level. For example, the following syntax creates a file called *99::pubs2.cmp*, which is compressed with the default compression level of 1:

```
dump database pubs2 to "compress::99::pubs2.cmp"
```

In general, the higher the compression numbers, the smaller your archives are compressed into. However, the compression result depends on the actual content of your files.

Table 27-6 shows the compression levels for the pubs2 database. These numbers are for reference only; the numbers for your site may differ depending on OS level and configuration.

Table 27-6: Compression levels and compressed file sizes for pub2

Compression levels	Compressed file size
No compression/Level 0	630K
Default compression/Level 1	128K
Level 2	124K
Level 3	121K
Level 4	116K
Level 5	113K
Level 6	112K
Level 7	111K
Level 8	110K
Level 9	109K

The higher the compression level, the more CPU-intensive the process is.

For example, you may not want to use a level-9 compression when archiving your files. Instead, consider the trade-off between processing effort and archive size. The default compression level (6) provides optimal CPU usage, producing an archive that is 60% to 80% smaller than a regular uncompressed archive. Sybase recommends that you initially use the default compression level, then increase or decrease the level based on your performance requirements.

For complete syntax information about dump database and dump transaction, see the *Reference Manual*.

Backup Server dump files and compressed dumps

When you perform dump database or dump transaction using an archive file that already exists, Backup Server automatically checks the header of the existing dump archive. If the header is unreadable, Backup Server assumes that the file is a valid non-archive file, and prompts you to change the dump archive with the following message:

```
Backup Server: 6.52.1.1: OPERATOR: Volume to be overwritten on
'/opt/SYBASE/DUMPS/model.dmp' has unrecognized label data.
Backup Server: 6.78.1.1: EXECUTE sp_volchanged
       @session_id = 5,
       @devname = '/opt/SYBASE/DUMPS/model.dmp',
       @action = { 'PROCEED' | 'RETRY' |
'ABORT' },
       @vname = <new_volume_name>
```

For this reason, if you perform dump database or dump transaction to a file without the compress:: option into an existing compressed dump archive, Backup Server does not recognize the header information of the archive because it is compressed.

Example

```
dump database model to 'compress::model.cmp'
go
dump database model to 'model.cmp'
go
```

The second dump database reports an error, and prompts you with sp_volchanged.

To prevent this error, either:

- Include the with init option in your subsequent dump database and dump transaction commands:

  ```
  dump database model to 'compress::model.cmp'
  go
  dump database model to 'model.cmp'
     with init
  go
  ```

- Include the compress:: option in subsequent dump database and dump transaction commands:

  ```
  dump database model to 'compress::model.cmp'
  go
  dump database model to 'compress::model.cmp'
  go
  ```

Using the compress:: option into uncompressed dump archives, as in this example, does not generate errors:

```
dump database model to ’model.cmp’
go
dump database model to ’compress::model.cmp’
go
```

Loading databases and transaction logs dumped with compress option

If you use dump ... compress to dump a database or transaction log, you must load this dump using the load ... compress option.

The partial syntax for load database .. compress and load transaction .. compress is:

load database *database_name*
from compress::*stripe_device*
…[stripe on compress::*stripe_device*]…

load transaction *database_name*
from compress::*stripe_device*
…[stripe on compress::*stripe_device*]…

The *database_name* in the syntax is the database you archived, and compress:: invokes the decompression of the archived database or transaction log. *archive_name* is the full path to the archived database or transaction log that you are loading. If you did not include a full path when you created your dump file, Adaptive Server created a dump file in the directory in which you started Adaptive Server.

Use the stripe on clause if you compressed the database or transaction log using multiple dump. See “Specifying additional dump devices: the stripe on clause” on page 851 for more information about the stripe on clause.

Note Do not use the *compression_level* variable for the load command.

Example

```
load database pubs2 from
   "compress::/opt/bin/Sybase/dumps/dmp090100.dmp"
```

```
Backup Server session id is:  19.  Use this value when executing the
’sp_volchanged’ system stored procedure after fulfilling any volume change
request from the Backup Server.
Backup Server: 4.132.1.1: Attempting to open byte stream device:
```

```
'compress::/opt/bin/Sybase/dumps/dmp090100.dmp::00'
Backup Server: 6.28.1.1: Dumpfile name 'pubs2002620A951  ' section number 1
mounted on byte stream 'compress::/opt/bin/Sybase/dumps/dmp090100.dmp::00'
Backup Server: 4.58.1.1: Database pubs2: 1382 kilobytes LOADed.
Backup Server: 4.58.1.1: Database pubs2: 3078 kilobytes LOADed.
Backup Server: 4.58.1.1: Database pubs2: 3086 kilobytes LOADed.
Backup Server: 3.42.1.1: LOAD is complete (database pubs2).
Use the ONLINE DATABASE command to bring this database online; SQL Server
will not bring it online automatically.
```

For complete syntax information about load database and load transaction, see the *Reference Manual*.

Specifying a remote Backup Server

Use the at *server_name* clause to send dump and load requests over the network to a Backup Server running on another machine.

Table 27-7 shows the syntax for dumping or loading from a remote Backup Server.

Table 27-7: Dumping to or loading from a remote Backup Server

	Backing up a database or log	Loading a database or log
	`dump {database | tran}` *database_name* `to [compress::[`*compression_level*`::]]` *stripe_device*	`load {database | tran}` *database_name* `from [compress::]`*stripe_device*
Remote Backup Server	`[at` *server_name*`]`	`[at` *server_name*`]`

	Backing up a database or log	Loading a database or log

	[density = *density*,
	blocksize = *number_bytes*,
	capacity = *number_kilobytes*,
	dumpvolume = *volume_name*,
	file = *file_name*]
	[stripe on
	[compress::[*compression_level*::]]
	stripe_device
	[at *server_name*]
	[density = *density*,
	blocksize = *number_bytes*,
	capacity = *number_kilobytes*,
	dumpvolume = *volume_name*,
	file = *file_name*] ...]
	[with{
	density = *density*,
	blocksize = *number_bytes*,
	capacity = *number_kilobytes*,
	dumpvolume = *volume_name*,
	file = *file_name*,
	[nodismount | dismount],
	[nounload | unload],
	retaindays = *number_days*,
	[noinit | init],
	[notify = {client |
	operator_console}]
	standby_access}]
		[density = *density*,
		dumpvolume = *volume_name,*
		file = *file_name*]
		[stripe
		[compress::]*stripe_device*
		[at *server_name*]
		[density = *density*,
		dumpvolume = *volume_name*,
		file = *file_name*] ...]
		[with{
		density = *density*,
		dumpvolume = *volume_name*,
		file = *file_name*,
		[nodismount | dismount],
		[nounload | unload],
		[notify = {client |
		operator_console}]
		standby_access}]

Sending dump and load requests over the network is ideal for installations that use a single machine with multiple tape devices for all backups and loads. Operators can be stationed at these machines, ready to service all tape change requests.

The following examples dump to and load from the remote Backup Server REMOTE_BKP_SERVER:

```
dump database pubs2 to "/dev/nrmt0" at REMOTE_BKP_SERVER
load database pubs2 from "/dev/nrmt0" at REMOTE_BKP_SERVER
```

The *server_name* must appear in the interfaces file on the computer where Adaptive Server is running, but does not need to appear in the sysservers table. The *server_name* must be the same in both the local and the remote interfaces file.

Specifying tape density, block size, and capacity

In most cases, the Backup Server uses a default tape density and block size that are optimal for your operating system; *we recommend that you use them*.

You can specify a density, block size, and capacity for each dump device. You can also specify the density, blocksize, and capacity options in the with clause for all dump devices. Characteristics that are specified for an individual tape device take precedence over those that you specify using the with clause.

Table 27-8 shows the syntax for specifying the tape density, block size, and capacity.

Table 27-8: Specifying tape density, block size, and capacity

	Backing up a database or log	**Loading a database or log**

	`dump {database | tran}`
	database_name
	`to [compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
		`load {database | tran}`
		database_name
		`from [compress::]`*stripe_device*
		`[at` *server_name*`]`

Characteristics of a single tape device	`[density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
		`[density =` *density*`,`

	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`]`
	`[stripe on`
	`[compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`] ...]`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`]`
		`[stripe on`
		`[compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`] ...]`

Characteristics of all dump devices	`[with{`
	`density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
		`[with{`
		`density =` *density*`,`

	Backing up a database or log	**Loading a database or log**
	`dumpvolume =` *volume_name*`,` `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `retaindays =` *number_days*`,` `[noinit | init],` `[notify = {client | operator_console}]` `standby_access}]`	`dumpvolume =` *volume_name*`,` `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `[notify = {client | operator_console}]` `standby_access}]`

The following sections provide greater detail about the density, blocksize, and capacity options.

Overriding the default density

The dump and load commands use the default tape density for your operating system. In most cases, this is the optimal density for tape dumps.

When you are dumping to tape on OpenVMS systems, you can override the default density with the density = *density* option. Valid densities are 800, 1600, 6250, 6666, 10000, and 38000. Not all densities are valid for all tape drives; specify a value that is correct for your drive.

This option has no effect on OpenVMS tape loads or on UNIX and PC platform dumps or loads.

Note Specify tape density only when using the init tape handling option. For more information on this option, see “Reinitializing a volume before a dump” on page 856.

Overriding the default block size

The blocksize parameter specifies the number of bytes per I/O operation for a dump device. By default, the dump and load commands choose the “best” block size for your operating system. Wherever possible, use these defaults.

You can use the blocksize = *number_bytes* option to override the default block size for a particular dump device. The block size must be at least one database page (2048 bytes) and must be an exact multiple of the database page size.

For OpenVMS systems, you can specify a block size only for dumps. Use a block size of less than or equal to 55,296.

For UNIX systems, the block size specified on the load command is ignored. Backup Server uses the block size that was used to make the dump.

Specifying a higher block size value

If you dump to a tape using the dump database or dump transaction commands, and specify a block size value which is higher than the maximum blocksize of a device as determined by Backup Server, then the dump or the load may fail on certain tape drives. An operating system error message displays; for example, on an 8mm tape drive on HP the error message is:

```
Backup Server: 4.141.2.22: [2] The 'write' call
failed for device 'xxx' with error number 22 (Invalid
argument). Refer to your operating system
documentation for further details.
```

You should not specify a block size greater than the device's block size stored in the tape device configuration file in *$SYBASE/backup_tape.cfg*. The block size for a device is the fifth field of the line in the tape device configuration file.

This error occurs only on tape drives where tape auto config is run; that is, the device models are not hard-coded in Backup Server code.

Specifying tape capacity for dump commands

By default, OpenVMS systems write until they reach the physical end-of-tape marker, and then signal that a volume change is required. For UNIX platforms that cannot reliably detect the end-of-tape marker, you must indicate how many kilobytes can be dumped to a tape.

If you specify the physical path name of the dump device, you must include the capacity = *number_kilobytes* parameter in the dump command. If you specify the logical dump device name, the Backup Server uses the *size* parameter stored in the sysdevices table, unless you override it with the capacity = *number_kilobytes* parameter.

The specified capacity must be at least five database pages (each page requires 2048 bytes). We recommend that you specify a capacity that is slightly below the capacity rated for your device.

A general rule for calculating capacity is to use 70 percent of the manufacturer's maximum capacity for the device, and allow 30 percent for overhead (inter-record gaps, tape marks, and so on). This rule works in most cases, but may not work in all cases because of differences in overhead across vendors and devices.

Non-rewinding tape functionality for Backup Server

The non-rewinding tape functionality automatically positions the tape at the end of valid dump data, which saves time when you want to perform multiple dump operations.

Dump label changes

Backup Server writes an End-of-File label, EOF3, at the end of every dump operation.

Note You cannot load a tape with this label into any version of Adaptive Server earlier then 12.0.

Tape operations

When a new dump is performed, Backup Server performs a scan for the last EOF3 label.

If the EOF3 label is found, the pertinent information is saved and the tape is positioned forward to the beginning of the next file on tape. This is the new append point.

If the EOF3 label is not found or any other problem is encountered, Backup Server rewinds the tape and scans forward. Any error that occurs during these steps does not abort the dump operation, but causes Backup Server to default to rewind-and-scan behavior. If the error persists during the rewind and scan, the dump command aborts.

Dump version compatibility

Backup Server activates non-rewinding logic only if the label version on the tape is greater than or equal to 5. Therefore, a dump command with the with init clause is needed to activate this logic. If a dump without init is initiated onto a volume with a label version less than 5, you are prompted to change the volume, and the dump starts on the next volume. The label version of a multi-volume dump does not change in the middle of the volume set.

Table 27-9 defines the label versions for which the new behavior is enabled.

Table 27-9: Label version compatibility

Label version	Enabled
‘3’	No
‘4’	No
‘5’	Yes
‘6’	Yes

Specifying the volume name

Use the with dumpvolume = *volume_name* option to specify the volume name. dump database and dump transaction write the volume name to the SQL tape label. load database and load transaction check the label. If the wrong volume is loaded, Backup Server generates an error message.

You can specify a volume name for each dump device. You can also specify a volume name in the with clause for all devices. Volume names specified for individual devices take precedence over those specified in the with clause.

Table 27-10 shows the syntax for specifying a volume name.

Table 27-10: Specifying the volume name

	Backing up a database or log	Loading a database or log
	`dump {database | tran}` *database_name* `to [compress::[`*compression_level*`::]]` *stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes*`,`	`load {database | tran}` *database_name* `from` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*`,`
Volume name for single device	`dumpvolume =` *volume_name*`,`	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`]` `[stripe on` `[compress::[`*compression_level*`::]]` *stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]` `[with{` `density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes*`,`	`file =` *file_name*`]` `[stripe on` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`]...]` `[with {` `density =` *density*`,`
Volume name for all devices	`dumpvolume =` *volume_name*`,`	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `retaindays =` *number_days*`,` `[noinit | init],` `[notify = {client` `|operator_console}]` `standby_access}]`	`file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `[notify = {client |` `operator_console}]` `standby_access}]`

Loading from a multifile volume

When you load a database dump from a volume that contains multiple dump files, specify the dump file name. If you omit the dump file name and specify only the database name, Backup Server loads the first dump file into the specified database. For example, entering the following command loads the first dump file from the tape into *pubs2*, regardless of whether that dump file contains data from *pubs2*:

```
load database pubs2 from "/dev/rdsk/clt3d0s6"
```

To avoid this problem, specify a unique dump file name each time you dump or load data. To get information about the dump files on a given tape, use the listonly = full option of load database.

Identifying a dump

When you dump a database or transaction log, Backup Server creates a default file name for the dump by concatenating the:

- Last 7 characters of the database name
- 2-digit year number
- 3-digit day of the year (1–366)
- Number of seconds since midnight, in hexadecimal

You can override this default using the file = *file_name* option. The file name cannot exceed 17 characters and must conform to the file naming conventions for your operating system.

You can specify a file name for each dump device. You can also specify a file name for all devices in the with clause. File names specified for individual devices take precedence over those specified in the with clause.

Table 27-11 shows the syntax for specifying the name of a dump.

Table 27-11: Specifying the file name for a dump

	Backing up a database or log	**Loading a database or log**

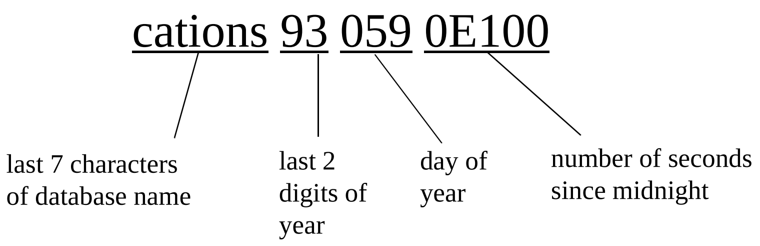
	`dump {database | tran}` *database_name*
	`to [compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dumpvolume =` *volume_name*`,`
		`load {database | tran}`
		database_name
		`from [compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*`,`

File name for single device	`file =` *file_name*`]`	`file =` *file_name*`]`

	Backing up a database or log	**Loading a database or log**
	`[stripe on` `[compress::[`*compression_level*`::]]` *stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]` `[with{` `density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes*`,` `dumpvolume =` *volume_name*`,`	`[stripe on` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]` `[with{` `density =` *density,* `dumpvolume =` *volume_name*`,`
File name for all devices	`file =` *file_name*`,`	`file =` *file_name*`,`
	`[nodismount | dismount],` `[nounload | unload],` `retaindays =` *number_days,* `[noinit | init],` `[notify = {client |` `operator_console}]` `standby_access}]`	`[nodismount | dismount],` `[nounload | unload],` `[notify = {client` `| operator_console}]` `standby_access}]`

The following examples dump the transaction log for the publications database without specifying a file name. The default file name, *cations930590E100*, identifies the database and the date and time the dump was made:

Figure 27-1: File-naming convention for database and transaction log dumps


cations 93 059 0E100
last 7 characters of database name
last 2 digits of year
day of year
number of seconds since midnight

Backup Server sends the file name to the default message destination or to the notify location for the dump command. Be sure to label each backup tape with the volume name and file name before storing it.

When you load a database or transaction log, you can use the file = *file_name* clause to specify which dump to load from a volume that contains multiple dumps.

When loading the dump from a multifile volume, you must specify the correct file name.

```
dump tran publications
    to "/dev/nrmt3"
load tran publications
    from "/dev/nrmt4"
    with file = "cations930590E100"
```

The following examples use a user-defined file-naming convention. The 15-character file name, *mydb97jul141800*, identifies the database (mydb), the date (July 14, 1997), and the time (18:00, or 6:00 p.m.) that the dump was made. Using the load command advances the tape to *mydb97jul141800* before loading:

```
dump database mydb
    to "/dev/nrmt3"
    with file = "mydb97jul141800"
load database mydb
    from "/dev/nrmt4"
    with file = "mydb97jul141800"
```

Improving dump or load performance

When you start Backup Server, you can use the -m parameter to improve the performance of the dump and load commands by configuring more shared memory for the Backup Server. The -m parameter specifies the maximum amount of shared memory used by the Backup Server. You must also configure your operating system to ensure that this amount of shared memory is available to the Backup Server. After a dump or load operation is completed, its shared memory segments are released.

Note Configuring more shared memory improves dump/load performance only if the performance limits of the hardware setup have not been reached. Increasing the value of -m may not result in improved performance when dumping to a slow tape device such as QIC, but it can improve performance significantly when dumping to a faster device, such as DLT.

Compatibility with prior versions

There are some compatibility issues between dump files and Backup Server. Table 27-12 indicates the dump file formats that can be loaded by the current and previous versions of local Backup Servers.

Table 27-12: Server for local operations

	New dump file format	**Old dump file format**
New version of server	Yes	Yes
Prior version of server	No	Yes

Table 27-13 and Table 27-14 indicate the dump file formats that can be loaded by the current and prior versions of remote Backup Servers. In a remote Backup Server scenario, the master server is the Backup Server on the same machine as the database and Adaptive Server Enterprise, and the slave server is the Backup Server on the same remote machine as the archive device.

Table 27-13 indicates the load operations that work when master server is the current version of Backup Server.

Table 27-13: New version of master server

	New dump file format	**Old dump file format**
New slave version of server	Yes	Yes

	New dump file format	Old dump file format
Prior slave version of server	No	Yes

Table 27-14 indicates the load operations that work when the master server is a prior version.

Table 27-14: Prior version of master server

	New dump file format	Old dump file format
New slave version of server	No	Yes
Prior slave version of server	No	Yes

Labels stored in integer format

Backup Server 12.0 and later store the stripe number in integer format. Earlier versions of Backup Server stored the 4-byte stripe number in the HDR1 label in ASCII format. These earlier versions of Backup Server cannot load a dump file that uses the newer dump format. However, Backup Server version 12.0 and later can read and write earlier versions of the dump format.

When performing a dump or load operation involving one or more remote servers, the operation aborts with an error message, if:

- The versions of one or more of the remote Backup Servers are earlier than 12.0, and the database is dumped to or loaded from more than 32 stripes.

  Or:

- The dump file from which one or more of the Backup Servers are reading during a load is from an earlier version's format, and the number of stripes from which the datbase is loaded is greater than 32.

Configuring system resources

Before you perform dumps and loads. you must configure the local and remote Backup Servers at startup by providing the appropriate values for the system resources controlled by the command line options. See the *Utility Guide* for a complete list of the command line options.

If your system resources are not configured properly, the dump or load can fail. For example, a remote dump to greater than 25 stripes with the local and remote Backup Servers booted with default configuration will fail because the maximum number of network connections that Backup Server can originate (specified by the -N option) is 25; however, by default the maximum number of server connections into the remote Backup Server (specified by the -C option) is 30.

To configure the system to use the higher stripe limitations, set the following operating system parameters:

- Number of shared memory segments to which a process can attach.
- Number of shared memory identifiers
- Swap space

If these parameters are not configured properly, when a dump is started to (or a load is started from) a large number of stripes, the operation may abort because of lack of system resources. In this case, you receive a message that Backup Server could not create or attach to a shared memory segment and therefore the SYBMULTBUF processes are terminated.

Setting shared memory usage

The syntax for starting Backup Server with the -m parameter is:

```
backupserver [-m nnn]
```

where *nnn* is the maximum amount of shared memory in megabytes that the Backup Server can use for all of its dump or load sessions.

The -m parameter sets the upper limit for shared memory usage. However, Backup Server may use less memory than specified if it detects that adding more memory will not improve performance.

Backup Server determines the amount of shared memory available for each stripe by dividing the -m value by the configured number of service threads (-P parameter).

The default value for -m is the number of service threads multiplied by 1MB. The default value for -P is 48, so the default maximum shared memory utilization is 48MB. However, Backup Server reaches this usage only if all the 48 service threads are active concurrently. The maximum value for -P is the maximum number of service threads, 12,288. (For more information about -P, see “Configuring user-defined roles” on page 358.)

The amount of shared memory per stripe available for Backup Server is inversely proportional to the number of service threads you allocate. If you increase the maximum number of service threads, you must increase the -m value, also, to maintain the same amount of shared memory per stripe. If you increase the -P value but do not increase the -m value, the shared memory allocated per stripe can decrease to the point that the dump or load cannot be processed.

To determine how much to increase the -m value, use this formula:

(-m value in MB) * 1024/(-P value)

If the value obtained by this formula is less than 128KB, Backup Server will not boot.

The minimum value for -m is 6MB. The maximum value for -m depends on operating system limits on shared memory.

If you create a dump using a Backup Server with a high shared memory value, and attempt to load the dump using a Backup Server with a lower shared memory value, Backup Server uses only the available memory. This results in degradation of the load performance.

If the amount of shared memory available per stripe at load time is less than twice the block size used at dump time, Backup Server aborts the load with an error message.

Setting maximum number of stripes

The maximum number of stripes that Backup Server can use is limited by the maximum number of Open Server threads it can create. Open Server imposes a maximum limit of 12k on the number of threads an application can create.

Backup Server creates one service thread for each stripe. Therefore, the maximum number of local stripes Backup Server can dump to or load from is 12,288.

As an additional limitation, Backup Server uses two file descriptors for each stripe, apart from the file descriptors associated with the error log file, interfaces file, and other system files. However, there is a per-thread limitation imposed by the operating system on the number of file descriptors. Open Server has a limitation of 1280 on the number of file descriptors that an application can keep track of.

The formula for determining the approximate maximum number of local stripes to which Backup Server can dump is:

$$\frac{\textbf{(The smaller of either the OS limitation or the OpenServer limitation) - 2}}{\textbf{2}}$$

The formula for determining the approximate maximum number of remote stripes to which Backup Server can dump is:

$$\frac{\textbf{(The smaller of either the OS limitation or the OpenServer limitation) - 2}}{\textbf{3}}$$

For details about the default and maximum file descriptor limits, see your operating system documentation.

Setting maximum number of network connections

The maximum number of network connections a local Backup Server can originate is limited by Open Server to 9118. Because of this, the maximum number of remote stripes that Backup Server can use in a single dump or load operation is 9118.

A remote Backup Server accepts a maximum of 4096 server connections at any one time. Therefore, the maximum number of remote stripes to a single remote Backup Server is 4096.

Setting maximum number of service threads

The -P parameter for Backup Server configures the number of service threads Open Server creates. The maximum number of service threads is 12,228. The minimum values is 6. The maximum number of threads equals the maximum number of stripes available. If you have started Backup Server without setting a high enough -P value, and you attempt to dump or load a database to a number of stripes that exceeds the number of threads, the dump or load operation fails.

Specifying additional dump devices: the *stripe on* clause

Striping allows you to use multiple dump devices for a single dump or load command. Use a separate stripe on clause to specify the name (and, if desired, the characteristics) of each device.

Each dump or load command can have multiple stripe on clauses.

Table 27-15 shows the syntax for using more than one dump device.

Table 27-15: Using more than one dump device

	Backing up a database or log	**Loading a database or log**
	`dump {database | tran}` *database_name* `to` `[compress::[`*compression_level*`::]]` *stripe_device* `[at` *server_name*`]` `[density =` *density,* `blocksize =` *number_bytes,* `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name*`,` `file =` *file_name*`]`	`load {database | tran}` *database_name* `from [compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name,* `file =` *file_name*`]`
Characteristics of an additional tape device (one set per device; up to 31 devices)	`[stripe on` `[compress::[`*compression_level*`::]]` *stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `blocksize =` *number_bytes,* `capacity =` *number_kilobytes,* `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]`	`[stripe on` `[compress::]`*stripe_device* `[at` *server_name*`]` `[density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`] ...]`
	`[with{` `density =` *density*`,` `blocksize =` *number_bytes*`,` `capacity =` *number_kilobytes*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `retaindays =` *number_days,* `[noinit | init],` `[notify = {client |` `operator_console}]` `standby_access}]`	`[with{` `density =` *density*`,` `dumpvolume =` *volume_name*`,` `file =` *file_name*`,` `[nodismount | dismount],` `[nounload | unload],` `[notify = {client |` `operator_console}]` `standby_access}]`

Dumping to multiple devices

The Backup Server divides the database into approximately equal portions and sends each portion to a different device. Dumps are made concurrently on all devices, reducing the time required to dump an individual database or transaction log. Because each tape stores only a portion of the database, it is less likely that a new tape will have to be mounted on a particular device.

Warning! Do not dump the master database to multiple tape devices. When loading the master database from tape or other removable media, you cannot change volumes unless you have another Adaptive Server that can respond to volume change messages.

Loading from multiple devices

You can use multiple devices to load a database or transaction log. Using multiple devices decreases both the time required for the load and the likelihood of having to mount multiple tapes on a particular device.

Using fewer devices to load than to dump

You can load a database or log even if one of your dump devices becomes unavailable between the dump and load. Specify fewer stripe clauses in the load command than you did in the dump command.

Note When you dump and load over the network, you must use the same number of drives for both operations.

The following examples use three devices to dump a database but only two to load it:

```
dump database pubs2 to "/dev/nrmt0"
    stripe on "/dev/nrmt1"
    stripe on "/dev/nrmt2"
load database pubs2 from "/dev/nrmt0"
    stripe on "/dev/nrmt1"
```

After the first two tapes are loaded, a message notifies the operator to load the third.

You can also dump a database to multiple operating system files. The following example is for Windows NT:

```
dump database pubs2 to "d:\backups\backup1.dat"
    stripe on "d:\backups\backup2.dat"
    stripe on "d:\backups\backup3.dat"
load database pubs2 from "/dev/nrmt0"
stripe on "d:\backups\backup2.dat"
    stripe on "d:\backups\backup3.dat"
```

Specifying the characteristics of individual devices

Use a separate at *server_name* clause for each stripe device attached to a remote Backup Server. If you do not specify a remote Backup Server name, the local Backup Server looks for the dump device on the local machine. If necessary, you can also specify separate tape device characteristics (density, blocksize, capacity, dumpvolume, and file) for individual stripe devices.

The following examples use three dump devices, each attached to the remote Backup Server REMOTE_BKP_SERVER.

On UNIX:

```
dump database pubs2
        to "/dev/nrmt0" at REMOTE_BKP_SERVER
        stripe on "/dev/nrmt1" at REMOTE_BKP_SERVER
        stripe on "/dev/nrmt2" at REMOTE_BKP_SERVER
```

On OpenVMS:

```
dump database pubs2
    to "MTA0:" at REMOTE_BKP_SERVER
    stripe on "MTA1:" at REMOTE_BKP_SERVER
    stripe on "MTA2:" at REMOTE_BKP_SERVER
```

Tape handling options

The tape handling options, which appear in the with clause, apply to all devices used for the dump or load. They include:

- nodismount to keep the tape available for additional dumps or loads

- unload to rewind and unload the tape following the dump or load
- retaindays to protect files from being overwritten
- init to reinitialize the tape rather than appending the dump files after the last end-of-tape mark

Table 27-16 shows the syntax for tape handling options.

Table 27-16: Tape handling options

	Backing up a database or log	**Loading a database or log**

	`dump {database | tran}` *database_name*
	`to [compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density,*
	`blocksize =` *number_bytes,*
	`capacity =` *number_kilobytes,*
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`]`
	`[stripe on`
	`[compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density,*
	`blocksize =` *number_bytes,*
	`capacity =` *number_kilobytes,*
	`dumpvolume =` *volume_name,*
	`file =` *file_name*`] ...]`
	`[with{`
	`density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`,`
		`load {database | tran}` *database_name*
		`from [compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*
		`file =` *file_name*`]`
		`[compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`] ...]`
		`[with{`
		`density =` *density*`,`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`,`

Tape Handling Options	`[nodismount | dismount],`
	`[nounload | unload],`
	`retaindays =` *number_days*`,`
	`[noinit | init],`
		`nodismount | dismount],`
		`[nounload | unload],`

	`[notify = {client |`
	`operator_console}]`
	`standby_access}]`
		`[notify = {client |`
		`operator_console}]`
		`standby_access}]`

Specifying whether to dismount the tape

On platforms that support logical dismounts, such as OpenVMS, tapes are dismounted when a dump or load completes. Use the nodismount option to keep the tape mounted and available for additional dumps or loads. This command has no effect on UNIX or PC systems.

Rewinding the tape

By default, both dump and load commands use the nounload tape handling option.

On UNIX systems, this prevents the tape from rewinding after the dump or load completes. This allows you to dump additional databases or logs to the same volume or to load additional databases or logs from that volume. Use the unload option for the last dump on the tape to rewind and unload the tape when the command completes.

On OpenVMS systems, tapes are always rewound after a dump or load completes. Use the unload option to unthread the tape and eject it from the drive. (This action is equivalent to the /UNLOAD qualifier for the OpenVMS DISMOUNT command.)

Protecting dump files from being overwritten

tape retention in days specifies the number of days that must elapse between the creation of a tape file and the time at which you can overwrite it with another dump. This server-wide variable, which you can is set with sp_configure, applies to all dumps requested from a single Adaptive Server.

Use the retaindays = *number_days* option to override the tape retention in days parameter for a single database or transaction log dump. The number of days must be a positive integer, or zero if the tape can be overwritten immediately.

Note tape retention in days and retaindays are meaningful only for disk, 1/4-inch cartridge, and single-file media. On multifile media, Backup Server checks only the expiration date of the first file.

Reinitializing a volume before a dump

By default, each dump is appended to the tape following the last end-of-tape mark. Tape volumes are not reinitialized. This allows you to dump multiple databases to a single volume. (New dumps can be appended only to the last volume of a multivolume dump.)

Use the init option to overwrite any existing contents of the tape. If you specify init, the Backup Server reinitializes the tape *without* checking for:

- ANSI access restrictions
- Files that have not yet expired
- Non-Sybase data ("foreign" tapes on OpenVMS)

The default, noinit, checks for all three conditions and sends a volume change prompt if any are present.

The following example initializes two devices, overwriting the existing contents with the new transaction log dumps:

```
dump transaction pubs2
    to "/dev/nrmt0"
    stripe on "/dev/nrmt1"
    with init
```

You can also use the init option to overwrite an existing file, if you are dumping a database to an operating system file. The following example is for Windows NT:

```
dump transaction pubs2
    to "d:\backups\backup1.dat"
    stripe on "d:\backups\backup2.dat"
    with init
```

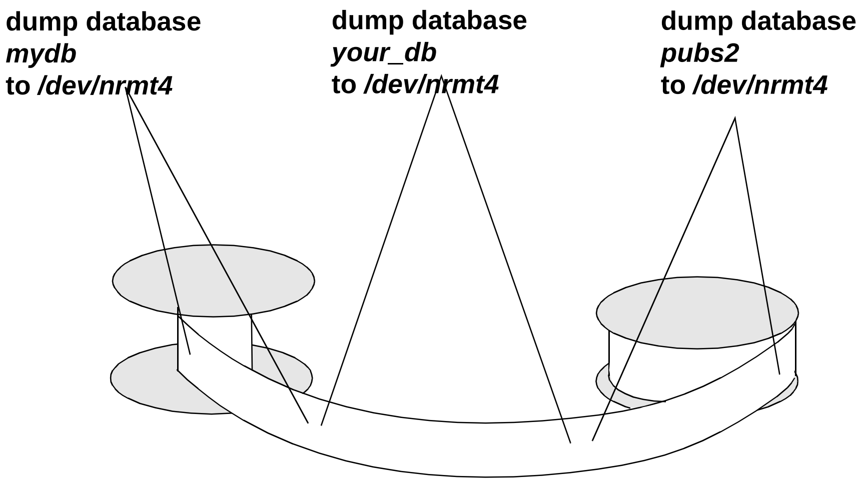
Dumping multiple databases to a single volume

To dump multiple databases to the same tape volume:

1. Use the init option for the first database. This overwrites any existing dumps and places the first dump at the beginning of the tape.
2. Use the default (noinit and nounload) option for subsequent databases. This places them one after the other on the tape.
3. Use the unload option for the last database on the tape. This rewinds and unloads the tape after you dump the last database.

Figure 27-2 illustrates how use to dump three databases to a single tape volume.

Figure 27-2: Dumping several databases to the same volume


dump database *mydb* to */dev/nrmt4*
dump database *your_db* to */dev/nrmt4*
dump database *pubs2* to */dev/nrmt4*

Overriding the default message destination

Backup Server messages inform the operator when to change tape volumes and how the dump or load is progressing. The default destination for these messages depends on whether the operating system offers an operator terminal feature.

The notify option, which appears in the with clause, allows you to override the default message destination for a dump or load. For this option to work, the controlling terminal or login session from which Backup Server was started must remain active for as long as Backup Server is working; otherwise, the sp_volchanged message is lost.

On operating systems that offer an operator terminal feature (such as Open VMS), volume change messages are always sent to an operator terminal on the machine where Backup Server is running. (OpenVMS routes messages to terminals that are enabled for TAPES, DISKS, or CENTRAL.) Use notify = client to route other Backup Server messages to the terminal session where the dump or load request initiated.

On systems such as UNIX that do not offer an operator terminal feature, messages are sent to the client that initiated the dump or load request. Use notify = operator_console to route messages to the terminal where the remote Backup Server was started.

Table 27-17 shows the syntax for overriding the default message destination.

Table 27-17: Overriding the default message destination

	Backing up a database or log	**Loading a database or log**

	`dump {database | tran}` *database_name*
	`to [compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density,*
	`blocksize =` *number_bytes,*
	`capacity =` *number_kilobytes,*
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`]`
	`[stripe on`
	`[compress::[`*compression_level*`::]]`
	stripe_device
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`blocksize =` *number_bytes,*
	`capacity =` *number_kilobytes,*
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`] ...]`
	`[with{`
	`density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`,`
	`[nodismount | dismount],`
	`[nounload | unload],`
	`retaindays =` *number_days*`,`
	`[noinit | init],`
		`load {database | tran}`
		database_name
		`from [compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*
		`file =` *file_name*`]`
		`[stripe on`
		`[compress::]`*stripe_device*
		`[at` *server_name*`]`
		`[density =` *density*`,`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`] ...]`
		`[with{`
		`density =` *density*`,`
		`dumpvolume =` *volume_name*`,`
		`file =` *file_name*`,`
		`[nodismount | dismount],`
		`[nounload | unload],`

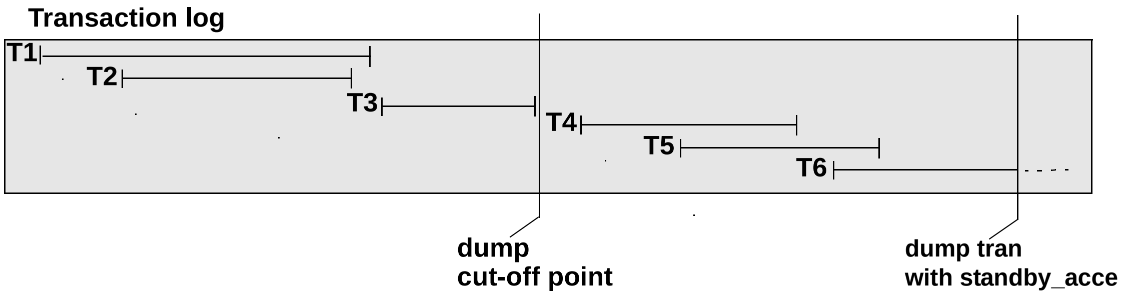
Message destination	`[notify = {client |`
	`operator_console}]`
		`[notify = {client |`
		`operator_console}]`

	`standby_access}]`	`standby_access}]`

Bringing databases online *with standby_access*

with standby_access causes dump transaction to dump only completed transactions. It dumps the transaction log up to the point at which there are no active transactions. If you do not use with standby_access, the entire transaction log, including records for all open transactions is dumped. A transaction log dump using with standby_access is illustrated in Figure 27-3.

Figure 27-3: Dump cut-off point for dump transaction with standby_access



In Figure 27-3, a dump transaction...with standby_access command is issued at a point where transactions T1 through T5 have completed and transaction T6 is still open. The dump cannot include T5 because T6 is still open, and it cannot include T4, because T5 is still open. Thus, the dump must stop at the end of transaction T3, where it will include completed transactions T1 through T3.

Syntax

The syntax for with standby_access is:

```
dump tran[saction] database_name to...
    [with standby_access]
```

For more information about the with dump tran...with standby_access option, see the *Adaptive Server Enterprise Reference Manual*.

When do I use *with standby_access*?

Use dump tran[saction]...with standby_access when you will be loading two or more transaction logs in sequence, and you want the database to be online between loads. For example, if you have a read-only database that gets its data by loading transaction dumps from a primary database. In this case, if the read-only database is used for generating a daily report based on transactions in the primary database, and the primary database's transaction log is dumped at the end of day, the daily cycle of operations is:

1. On the primary database: dump tran[saction]...with standby_access
2. On the read-only database: load tran[saction]...
3. On the read-only database: online database for standby_access

Warning! If a transaction log contains open transactions, and you dump it without using with standby_access, Adaptive Server will not allow you to load the log, bring the database online, and then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after loading a dump originally made with standby_access or after loading the entire series.

Bring databases online *with standby_access*

The online database command also includes a with standby_access option. Use for standby_access to bring a database online after loading it with a dump that was made using the with standby_access option.

Warning! If you try to use online database for standby_access with a transaction log that was not dumped using the with standby_access option, the command will fail.

Syntax

The syntax for online database is:

online database *database_name* [for standby_access]

For more information about the with online database...for standby_access option, see the Adaptive Server Enterprise Reference Manual.

Getting information about dump files

If you are unsure of the contents of a tape, use the with headeronly or with listonly option of the load commands to request that information.

Table 27-18 shows the syntax for finding the contents of a tape.

Table 27-18: Listing dump headers or file names

	Listing information about a dump

	`load {database | tran}` *database_name*
	`from [compress::]`*stripe_device*
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`dumpvolume =` *volume_name*
	`file =` *file_name*`]`
	`[stripe on [compress::]`*stripe_device*
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`] ...]`
	`[with{`
	`density =` *density*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`,`
	`[nodismount | dismount],`
	`[nounload | unload],`

List header information
List files on tape
	`[headeronly [, file =` *filename* `]],`
	`[listonly [= full]],`

	`[notify = {client | operator_console}]`
	`standby_access}]`

Note Neither with headeronly nor with listonly loads the dump files after displaying the report.

Requesting dump header information

with headeronly returns the header information for a single file. If you do not specify a file name, with headeronly returns information about the first file on the tape.

The header indicates whether the dump is for a database or transaction log, the database ID, the file name, and the date the dump was made. For database dumps, it also shows the character set, sort order, page count, and next object ID. For transaction log dumps, it shows the checkpoint location in the log, the location of the oldest begin transaction record, and the old and new sequence dates.

The following example returns header information for the first file on the tape and then for the file *mydb9229510945*:

```
load database mydb
    from "/dev/nrmt4"
    with headeronly
load database mydb
    from "/dev/nrmt4"
    with headeronly, file = "mydb9229510945"
```

Here is sample output from headeronly:

```
Backup Server session id is: 44. Use this value when executing the
‘sp_volchanged’ system stored procedure after fulfilling any volume change
request from the Backup Server.
Backup Server: 4.28.1.1: Dumpfile name ‘mydb9232610BC8 ‘ section number 0001
mounted on device ‘backup/SQL_SERVER/mydb.db.dump’
This is a database dump of database ID 5 from Nov 21 1992 7:02PM.
Database contains 1536 pages; checkpoint RID=(Rid pageid = 0x404; row num =
0xa); next object ID=3031; sort order ID=50, status=0; charset ID=1.
```

Determining the database, device, file name, and date

with listonly returns a brief description of each dump file on a volume. It includes the name of the database, the device used to make the dump, the file name, the date and time the dump was made, and the date and time it can be overwritten. with listonly = full provides greater detail. Both reports are sorted by SQL tape label.

Following is sample output of a load database command with listonly:

```
Backup Server: 4.36.1.1: Device ‘/dev/nrst0’:
File name: ‘model9320715138  ‘
Create date & time: Monday, Jul 26, 1993, 23:58:48
Expiration date & time:  Monday, Jul 26, 1993, 00:00:00
Database name: ‘model                           ‘
```

and sample output from with listonly = full:

```
Backup Server: 4.37.1.1: Device ‘/dev/nrst0’:
```

```
Label id: ‘HDR1’
File name:‘model9320715138  ‘
Stripe count:0001
Device typecount:01
Archive volume number:0001
Stripe position:0000
Generation number:0001
Generation version:00
Create date & time:Monday, Jul 26, 1993, 23:58:48
Expiration date & time:Monday, Jul 26, 1993, 00:00:00
Access code:‘ ‘
File block count:000000
Sybase id string:
‘Sybase  ‘Reserved:‘       ‘
Backup Server: 4.38.1.1: Device ‘/dev/nrst0’:
Label id:‘HDR2’
Record format:‘F’
Max. bytes/block:55296
Record length:02048
Backup format version:01
Reserved:‘   ‘
Database name:‘model                         ‘
Buffer offset length:00
Reserved:‘                            ‘
```

After listing all files on a volume, the Backup Server sends a volume change request:

```
Backup Server: 6.30.1.2: Device /dev/nrst0: Volume cataloguing
complete.
Backup Server: 6.51.1.1: OPERATOR: Mount the next volume to search.
Backup Server: 6.78.1.1: EXECUTE sp_volchanged
          @session_id = 5,
          @devname = ‘/dev/nrst0’,
          @action = { ‘PROCEED’ | ‘RETRY’ | ‘ABORT’ },
          @fname = ‘
```

The operator can use sp_volchanged to mount another volume and signal the volume change or to terminate the search operation for all stripe devices.

Copying the log after a device failure

Normally, dump transaction truncates the inactive portion of the log after copying it. Use with no_truncate to copy the log without truncating it.

no_truncate allows you to copy the transaction log after failure of the device that holds your data. It uses pointers in the sysdatabases and sysindexes tables to determine the physical location of the transaction log. It can be used only if your transaction log is on a separate segment and your master database is accessible.

Warning! Use no_truncate only if media failure makes your data segment inaccessible. Never use no_truncate on a database that is in use.

Copying the log with no_truncate is the first step described in "Recovering a database: step-by-step instructions" on page 874.

Table 27-19 shows the syntax for copying a log after a device failure.

Table 27-19: Copying the log file after a device failure

	Copying with the *no_truncate* option

	`dump transaction` *database_name*
	`to [compress::[`*compression_level*`::]]` *stripe_device*
	`[at` *server_name*`]`
	`[density =` *density,*
	`blocksize =` *number_bytes,*
	`capacity =` *number_kilobytes,*
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`]`
	`[stripe on [compress::[`*compression_level*`::]]` *stripe_device*
	`[at` *server_name*`]`
	`[density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dump volume =` *volume_name*`,`
	`file =` *file_name*`] ...]`
	`[with{`
	`density =` *density*`,`
	`blocksize =` *number_bytes*`,`
	`capacity =` *number_kilobytes*`,`
	`dumpvolume =` *volume_name*`,`
	`file =` *file_name*`,`
	`[nodismount | dismount],`
	`[nounload | unload],`
	`retaindays =` *number_days*`,`
	`[noinit | init],`

Do not truncate log	`no_truncate,`

	`[notify = {client | operator_console}]`
	`standby_access}]`

You can use no_truncate with striped dumps, tape initialization, and remote Backup Servers. Here is an example:

```
dump transaction mydb
to "/dev/nrmt0" at REMOTE_BKP_SERVER
with init, no_truncate,
notify = "operator_console"
```

Truncating a log that is not on a separate segment

If a database does not have a log segment on a separate device from data segments, you cannot use dump transaction to copy the log and then truncate it. For these databases, you must:

1. Use the special with truncate_only option of dump transaction to truncate the log so that it does not run out of space
2. Use dump database to copy the entire database, including the log

Because it doesn't copy any data, with truncate_only requires only the name of the database:

```
dump transaction database_name with truncate_only
```

The following example dumps the database mydb, which does not have a log segment on a separate device from data segments, and then truncates the log:

```
dump database mydb to mydevice
dump transaction mydb with truncate_only
```

Truncating the log in early development environments

In early development environments, the transaction log is quickly filled by creating, dropping, and re-creating stored procedures and triggers and checking integrity constraints. Recovery of data may be less important than ensuring that there is adequate space on database devices.

with truncate_only allows you to truncate the transaction log without making a backup copy:

```
dump transaction database_name with truncate_only
```

After you run dump transaction with truncate_only, you must dump the database before you can run a routine log dump.

Truncating a log that has no free space

When the transaction log is very full, you may not be able to use your usual method to dump it. If you used dump transaction or dump transaction with truncate_only, and the command failed because of insufficient log space, use the special with no_log option of dump transaction:

```
dump transaction database_name with no_log
```

This option truncates the log without logging the dump transaction event. Because it doesn't copy any data, it requires only the name of the database.

Warning! Use dump transaction with no_log as a last resort, and use it only once after dump transaction with truncate_only fails. If you continue to load data after entering dump transaction with no_log, you may fill the log completely, causing any further dump transaction commands to fail. Use alter database to allocate additional space to the database.

All occurrences of dump tran with no_log are reported in the Adaptive Server error log. The message includes the user ID of the user executing the command. Messages indicating success or failure are also sent to the error log. no_log is the only dump option that generates error log messages.

Dangers of using *with truncate_only* and *with no_log*

with truncate_only and with no_log allow you to truncate a log that has become disastrously short of free space. Neither option provides a means to recover transactions that have committed since the last routine dump.

Warning! Run dump database at the earliest opportunity to ensure that your data can be recovered.

The following example truncates the transaction log for mydb and then dumps the database:

```
dump transaction mydb
    with no_log
dump database mydb to ...
```

Providing enough log space

Every use of dump transaction...with no_log is considered an error and is recorded in the server's error log. If you have created your databases with log segments on a separate device from data segments, written a last-chance threshold procedure that dumps your transaction log often enough, and allocated enough space to your log and database, you should not have to use this option.

However, some situations can still cause the transaction log to become too full, even with frequent log dumps. The dump transaction command truncates the log by removing all pages from the beginning of the log, up to the page preceding the page that contains an uncommitted transaction record (known as the oldest active transaction). The longer this active transaction remains uncommitted, the less space is available in the transaction log, since dump transaction cannot truncate additional pages.

This can happen when applications with very long transactions modify tables in a database with a small transaction log, which indicates you should increase the size of the log. It also occurs when transactions inadvertently remain uncommitted for long periods of time, such as when an implicit begin transaction uses the chained transaction mode or when a user forgets to complete the transaction. You can determine the oldest active transaction in each database by querying the syslogshold system table.

The *syslogshold* table

The syslogshold table is in the master database. Each row in the table represents either:

- The oldest active transaction in a database, or
- The Replication Server truncation point for the database's log.

A database may have no rows in syslogshold, a row representing one of the above, or two rows representing both of the above. For information about how a Replication Sever truncation point affects the truncation of the database's transaction log, see the Replication Server documentation.

Querying syslogshold provides a "snapshot" of the current situation in each database. Since most transactions last for only a short time, the query's results may not be consistent. For example, the oldest active transaction described in the first row of syslogshold may finish before Adaptive Server completes the query of syslogshold. However, when several queries of syslogshold over time query the same row for a database, that transaction may prevent a dump transaction from truncating any log space.

When the transaction log reaches the last-chance threshold, and dump transaction cannot free up space in the log, you can query syslogshold and sysindexes to identify the transaction holding up the truncation. For example:

```
select H.spid, H.name
from master..syslogshold H, threshdb..sysindexes I
where H.dbid = db_id("threshdb")
and I.id = 8
and H.page = I.first
spid    name
------  -------------------------------------
     8  $user_transaction

(1 row affected)
```

This query uses the object ID associated with syslogs (8) in the threshdb database to match the first page of its transaction log with the first page of the oldest active transaction in syslogshold.

You can also query syslogshold and sysprocesses in the master database to identify the specific host and application owning the oldest active transactions. For example:

```
select P.hostname, P.hostprocess, P.program_name,
   H.name, H.starttime
from sysprocesses P, syslogshold H
where P.spid = H.spid
and H.spid != 0
hostname hostprocess program_name name              starttime
-------- ----------- ----------- ----------------- ------------------
eagle          15826 isql        $user_transaction  Sep  6 1997 4:29PM
hawk           15859 isql        $user_transaction  Sep  6 1997 5:00PM
condor         15866 isql        $user_transaction  Sep  6 1997 5:08PM

(3 rows affected)
```

Using the above information, you can notify or kill the user process owning the oldest active transaction and proceed with the dump transaction. You can also include the above types of queries in the threshold procedures for the database as an automatic alert mechanism. For example, you may decide that the transaction log should never reach its last-chance threshold. If it does, your last-chance threshold procedure (sp_thresholdaction) alerts you with information about the oldest active transaction preventing the transaction dump.

Note The initial log records for a transaction may reside in a user log cache, which is not visible in syslogshold until the records are flushed to the log (for example, after a checkpoint).

For more information about the syslogshold system table, see the *Adaptive Server Enterprise Reference Manual*. For information about the last-chance threshold and threshold procedures, see Chapter 29, "Managing Free Space with Thresholds."

Responding to volume change requests

On UNIX and PC systems, use sp_volchanged to notify the Backup Server when the correct volumes have been mounted. On OpenVMS systems, use the REPLY command.

To use sp_volchanged, log in to any Adaptive Server that can communicate with both the Backup Server that issued the volume change request and the Adaptive Server that initiated the dump or load.

sp_volchanged syntax

Use this syntax for sp_volchanged:

```
sp_volchanged session_id, devname, action
     [ ,fname  [, vname ] ]
```

- Use the *session_id* and *devname* parameters specified in the volume change request.
- *action* specifies whether to abort, proceed with, or retry the dump or load.

- *fname* specifies the file to load. If you do not specify a file name, Backup Server loads the file = *file_name* parameter of the load command. If neither sp_volchanged nor the load command specifies which file to load, the Backup Server loads the first file on the tape.
- The Backup Server writes the *vname* in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. During loads, the Backup Server uses the *vname* to confirm that the correct tape has been mounted. If you do not specify a *vname*, the Backup Server uses the volume name specified in the dump or load command. If neither sp_volchanged nor the command specifies a volume name, the Backup Server does not check this field in the ANSI tape label.

Volume change prompts for dumps

This section describes the volume change prompts that appear while you are dumping a database or transaction log. Each prompt includes the possible operator actions and the appropriate sp_volchanged response.

- `Mount the next volume to search.`

  When appending a dump to an existing volume, the Backup Server issues this message if it cannot find the end-of-file mark.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *devname*, `abort`
Mount a new volume and proceed with the dump	`sp_volchanged` *session_id*, *devname*, `proceed [,` *fname* `[,` *vname*`]]`

- `Mount the next volume to write.`

  The Backup Server issues this message when it reaches the end of the tape. This occurs when it detects the end-of-tape mark, dumps the number of kilobytes specified by the capacity parameter of the dump command, or dumps the *high* value specified for the device in the sysdevices system table.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *devname*, `abort`
Mount the next volume and proceed with the dump	`sp_volchanged` *session_id*, *devname*, `proceed [,` *fname* `[,` *vname*`]]`

- `Volume on device devname has restricted access (code access_code).`

  Dumps that specify the init option overwrite any existing contents of the tape. Backup Server issues this message if you try to dump to a tape with ANSI access restrictions without specifying the init option.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *devname*, `abort`
Mount another volume and retry the dump	`sp_volchanged` *session_id*, *devname*, `retry [,` *fname* `[,` *vname*`]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged` *session_id*, *devname*, `proceed [,` *fname* `[,` *vname*`]]`

- `Volume on device devname is expired and will be overwritten.`

  Dumps that specify the init option overwrite any existing contents of the tape. During dumps to single-file media, Backup Server issues this message if you have not specified the init option and the tape contains a dump whose expiration date has passed.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *devname*, `abort`
Mount another volume and retry the dump	`sp_volchanged` *session_id*, *session_id*, `retry [,` *session_id* `[,` *session_id*`]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged` *session_id*, *session_id*, `proceed [,` *session_id* `[,` *session_id*`]]`

- `Volume to be overwritten on ’devname’ has not expired: creation date on this volume is creation_date, expiration date is expiration_date.`

  On single-file media, the Backup Server checks the expiration date of any existing dump unless you specify the init option. The Backup Server issues this message if the dump has not yet expired.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *session_id*, `abort`
Mount another volume and retry the dump	`sp_volchanged` *session_id*, *session_id*, `retry [,` *session_id* `[,` *session_id*`]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged` *session_id*, *session_id*, `proceed [,` *session_id* `[,` *session_id*`]]`

- `Volume to be overwritten on ’devname’ has unrecognized label data.`

  Dumps that specify the init option overwrite any existing contents of the tape. Backup Server issues this message if you try to dump to a new tape or a tape with non-Sybase data without specifying the init option.

The operator can	By replying
Abort the dump	`sp_volchanged` *session_id*, *session_id*, `abort`
Mount another volume and retry the dump	`sp_volchanged` *session_id*, *session_id*, `retry` [, *session_id* [, *session_id*]]
Proceed with the dump, overwriting any existing contents	`sp_volchanged` *session_id*, *session_id*, `proceed` [, *session_id* [, *session_id*]]

Volume change prompts for loads

Following are the volume change prompts and possible operator actions during loads:

- `Dumpfile ’fname’ section vname found instead of ’fname’ section vname.`

  The Backup Server issues this message if it cannot find the specified file on a single-file medium.

The operator can	By replying
Abort the load	`sp_volchanged` *session_id*, *session_id*, `abort`
Mount another volume and try to load it	`sp_volchanged` *session_id*, *session_id*, `retry` [, *session_id* [, *session_id*]]
Load the file on the currently mounted volume, even though it is not the specified file (not recommended)	`sp_volchanged` *session_id*, *session_id*, `proceed` [, *session_id* [, *session_id*]]

- `Mount the next volume to read.`

  The Backup Server issues this message when it is ready to read the next section of the dump file from a multivolume dump.

The operator can	By replying
Abort the load	`sp_volchanged` *session_id*, *session_id*, `abort`
Mount the next volume and proceed with the load	`sp_volchanged` *session_id*, *session_id*, `proceed` [, *session_id* [, *session_id*]]

- `Mount the next volume to search.`

  The Backup Server issues this message if it cannot find the specified file on multifile medium.

The operator can	By replying
Abort the load	`sp_volchanged` *session_id*, *session_id*, `abort`
Mount another volume and proceed with the load	`sp_volchanged` *session_id*, *session_id*, `proceed` `[,` *session_id* `[,` *session_id*`]]`

Recovering a database: step-by-step instructions

The symptoms of media failure are as variable as the causes. If only a single block on the disk is bad, your database may appear to function perfectly for some time after the corruption occurs, unless you are running dbcc commands frequently. If an entire disk or disk controller is bad, you will not be able to use a database. Adaptive Server marks the database as suspect and displays a warning message. If the disk storing the master database fails, users will not be able to log in to the server, and users already logged in will not be able to perform any actions that access the system tables in master.

When your database device fails, Sybase recommends the following steps:

1. Get a current log dump of *every database on the device*.
2. Examine the space usage of *every database on the device*.
3. After you have gathered this information for all databases on the device, drop each database.
4. Drop the failed device.
5. Initialize new devices.
6. Re-create the databases, one at a time.
7. Load the most recent database dump into each database.
8. Apply each transaction log dump in the order in which it was created.

These steps are described in detail in the following sections.

Getting a current dump of the transaction log

Use dump transaction with no_truncate to get a current transaction log dump *for each database on the failed device*. For example, to get a current transaction log dump of mydb:

```
dump transaction mydb
to "/dev/nrmt0" at REMOTE_BKP_SERVER
with init, no_truncate,
notify = "operator_console"
```

Examining the space usage

The following steps are recommended to determine which devices your database uses, how much space is allocated on each device, and whether the space is used for data, log, or both. You can use this information when re-creating your databases to ensure that the log, data, and indexes reside on separate devices, and to preserve the scope of any user segments you have created.

Note You can also use these steps to preserve segment mappings when moving a database dump from one server to another (on the same hardware and software platform).

If you do not use this information to re-create the device allocations for damaged databases, Adaptive Server will *remap* the sysusages table after load database to account for discrepancies. This means that the database's system-defined and user-defined segments no longer match the appropriate device allocations. Incorrect information in sysusages can result in the log being stored on the same devices as the data, even if the data and the log were separate before recovery. It can also change user-defined segments in unpredictable ways, and can result in a database that cannot be created using a standard create database command.

To examine and record the device allocations for all damaged databases:

1 In master, examine the device allocations and uses for the damaged database:

```
select segmap, size from sysusages
    where dbid = db_id("database_name")
```

2 Examine the output of the query. Each row with a segmap of "3" represents a data allocation; each row with a segmap of "4" represents a log allocation. Higher values indicate user-defined segments; treat these as data allocations, to preserve the scope of these segments. The size column indicates the number of blocks of data. Note the order, use, and size of each disk piece.

For example, this output from a server that uses 2K logical pages:

```
segmap        size
-------       --------
     3          10240
     3           5120
     4           5120
     8           1024
     4           2048
```

translates into the sizes and uses described in Table 27-20.

Table 27-20: Sample device allocation

Device allocation	Megabytes
Data	20
Data	10
Log	10
Data (user-defined segment)	2
Log	4

Note If the segmap column contains 7s, your data and log are on the same device, and you can recover only up to the point of the most recent database dump. *Do not* use the log on option to create database. Just be sure that you allocate as much (or more) space than the total reported from sysusages.

3 Run sp_helpdb *database_name* for the database. This query lists the devices on which the data and logs are located:

```
name        db_size owner   dbid    created
-------     ------- ------  ----    -----------
mydb        46.0 MB sa        15    Apr  9 1991

status          device_fragments    size          usage
--------------  ----------------    -----         ------------
no options set  datadev1            20 MB         data only
                datadev2            10 MB         data only
```

```
datadev3                2 MB          data only
logdev1                10 MB          log only
 logdev1                 4 MB          log only
```

Dropping the databases

After you have performed the preceding steps *for all databases on the failed device*, use drop database to drop each database.

Note If tables in other databases contain references to any tables in the database you are trying to drop, you must remove the referential integrity constraints with alter table before you can drop the database.

If the system reports errors because the database is damaged when you issue drop database, use the dropdb option of the dbcc dbrepair command:

```
dbcc dbrepair (mydb, dropdb)
```

See the *Troubleshooting Guide* for more information about dbcc dbrepair.

Dropping the failed devices

After you have dropped each database, use sp_dropdevice to drop the failed device. See the *Adaptive Server Enterprise Reference Manual* for more information.

Initializing new devices

Use disk init to initialize the new database devices. See Chapter 16, “Initializing Database Devices,” for more information.

Re-creating the databases

Use the following steps to re-create each database using the segment information you collected earlier.

Note If you chose not to gather information about segment usage, use create database...for load to create a new database that is at least as large as the original.

1 Use create database with the for load option. Duplicate all device fragment mappings and sizes for each row of your database from the sysusages table, *up to and including the first log device*. Use the order of the rows as they appear in sysusages. (The results of sp_helpdb are in alphabetical order by device name, not in order of allocation.) For example, to re-create the mydb database allocations shown in Table 27-20 on page 876, enter:

```
create database mydb
   on datadev1 = 20,
      datadev2 = 10
log on logdev1 = 10
for load
```

Note create database...for load temporarily locks users out of the newly created database, and load database marks the database offline for general use. This prevents users from performing logged transactions during recovery.

2 Use alter database with the for load option to re-create the remaining entries, in order. Remember to treat device allocations for user segments as you would data allocations.

In this example, to allocate more data space on datadev3 and more log space on logdev1, the command is:

```
alter database mydb
   on datadev3 = “2M”
log on logdev1= ”4M”
for load
```

Loading the database

Reload the database using load database. If the original database stored objects on user-defined segments (sysusages reports a segmap greater than 7) and your new device allocations match those of the dumped database, Adaptive Server preserves the user segment mappings.

If you did not create the new device allocations to match those of the dumped database, Adaptive Server will remap segments to the available device allocations. This remapping may also mix log and data on the same physical device.

Note If an additional failure occurs while a database is being loaded, Adaptive Server does not recover the partially loaded database, and notifies the user. You must restart the database load by repeating the load command.

Loading the transaction logs

Use load transaction to apply transaction log backups *in the same sequence in which they were made*.

Adaptive Server checks the timestamps on each dumped database and transaction log. If the dumps are loaded in the wrong order, or if user transactions have modified the transaction log between loads, the load fails.

If you dumped the transaction log using with standby_access, you must also load the database using standby_access.

After you have brought a database up to date, use dbcc commands to check its consistency.

Loading a transaction log to a point in time

You can recover a database up to a specified point in time in its transaction log. To do so, use the until_time option of load transaction. This is useful if, for example, a user inadvertently drops an important table; you can use until_time to recover the changes made to the database containing the table up to a time just before the table was dropped.

To use until_time effectively after data has been destroyed, you must know the exact time the error occurred. You can find this by issuing a select getdate at the time of the error. For example, suppose a user accidentally drops an important table, and then a few minutes later you get the current time in milliseconds:

```
select convert(char(26), getdate(), 109)
--------------------------
Mar 26 1997 12:45:59:650PM
```

After dumping the transaction log containing the error and loading the most recent database dump, load the transaction logs that were created after the database was last dumped. Then, load the transaction log containing the error by using until_time; for example:

```
load transaction employees_db
from "/dev/nrmt5"
with until_time = "Mar 26 1997 12:35:59: 650PM"
```

After you load a transaction log using until_time, Adaptive Server restarts the database's log sequence. This means that until you dump the database again, you cannot load subsequent transaction logs after the load transaction using until_time. You will need to dump the database before you can dump another transaction log.

Bringing the databases online

In this example, the transaction log is loaded up to a time just before the table drop occurred. After you have applied all transaction log dumps to a database, use online database to make it available for use. In this example, the command to bring the mydb database online is:

```
online database mydb
```

Replicated databases

Before you upgrade replicated databases to the current version of Adaptive Server, the databases must be online. However, you cannot bring replicated databases online until the logs are drained. If you try to bring a replicated database online before the logs are drained, Adaptive Server issues the following message:

```
Database is replicated, but the log is not yet
drained. This database will come online
automatically after the log is drained.
```

When Replication Server, via the Log Transfer Manager (LTM), drains the log, online database is automatically issued.

Upgrading to the current release of Adaptive Server

Refer to the installation documentation for your platform for upgrade instructions for Adaptive Server users that have replicated databases.

Load sequence

The load sequence for loading replicated databases is: load database, replicate, load transaction, replicate, and so on. At the end of the load sequence, issue online database to bring the databases online. Databases that are offline because they are in a load sequence are not automatically brought online by Replication Server.

Warning! Do not issue online database until all transaction logs are loaded.

Loading database dumps from older versions

When you upgrade an Adaptive Server installation is upgraded to a new release, all databases associated with that server are automatically upgraded.

As a result, database and transaction log dumps created with a previous version of Adaptive Server must be upgraded before they can be used with the current version of Adaptive Server.

Adaptive Server provides an automatic upgrade mechanism – on a per-database basis – for upgrading a database or transaction log made with Backup Sever to the current Adaptive Server release, thus making the dump compatible for use. This mechanism is entirely internal to Adaptive Server, and requires no external programs. It provides the flexibility of upgrading individual dumps as needed.

The following tasks are not supported by this automatic upgrade functionality:

- Loading an older release of the master database. That is, if you upgraded Adaptive Server to the current version, you cannot load a dump of the master database from which you upgraded.
- Installing new or modified stored procedures. Continue to use installmaster.

- Loading and upgrading dumps generated previous to SQL Server release 10.0.

How to upgrade a dump to Adaptive Server

To upgrade a user database or transaction log dump to the current release of Adaptive Server:

1. Use load database and load transaction to load the dump to be upgraded.

   Adaptive Server determines from the dump header which version it is loading. After the dump header is read, and before Backup Server begins the load, the database is marked offline by load database or load transaction. This makes the database unavailable for general use (queries and use *database* are not permitted), provides the user greater control over load sequences, and eliminates the possibility that other users will accidentally interrupt a load sequence.

2. Use online database, after the dump has successfully loaded, to activate the upgrade process.

   Note Do *not* issue online database until after all transaction dumps are loaded.

   Prior to SQL Server version 11.0, a database was automatically available at the end of a successful load sequence. With the current version of Adaptive Server, the user is required to bring the database online after a successful load sequence, using online database.

   For dumps loaded from SQL Server version 10.0, online database activates the upgrade process to upgrade the dumps just loaded. After the upgrade is successfully completed, Adaptive Server places the database online and the database is ready for use.

   For dumps loaded from the current version of Adaptive Server, no upgrade process is activated. You must still issue online database to place the database online – load database marks it as offline.)

   Each upgrade step produces a message stating what it is about to do.

   An upgrade failure leaves the database offline and produces a message stating that the upgrade failed and the user must correct the failure.

For more information about online database, see the *Adaptive Server Enterprise Reference Manual*.

3 After successful execution of online database, use dump database. The database must be dumped before a dump transaction is permitted. A dump transaction on a newly created or upgraded database is not permitted until a successful dump database has occurred.

The *database offline* status bit

The "database offline" status bit indicates that the database is not available for general use. You can determine whether a database is offline by using sp_helpdb. It will show that the database is offline if this bit is set.

When a database is marked offline by load database, a status bit in the sysdatabases table is set and remains set until the successful completion of online database.

The "database offline" status bit works in combination with any existing status bits. It augments the following status bit to provide additional control:

- In recovery

The "database offline" status bit overrides the following status bits:

- DBO use only
- Read only

The following status bits override the "database offline" status bit:

- Began upgrade
- Bypass recovery
- In load
- Not recovered
- Suspect
- Use not recovered

Although the database is not available for general use, you can user these commands when the database is offline:

- dump database and dump transaction
- load database and load transaction

- alter database on device
- drop database
- online database
- dbcc diagnostics (subject to dbcc restrictions)

Version identifiers

The automatic upgrade feature provides version identifiers for Adaptive Server, databases, and log record formats:

- Configuration upgrade version ID – shows the current version of Adaptive Server; it is stored in the sysconfigures table. sp_configure displays the current version of Adaptive Server as “upgrade version.”
- Upgrade version indicator – shows the current version of a database and is stored in the database and dump headers. The Adaptive Server recovery mechanism uses this value to determine whether the database should be upgraded before being made available for general use.
- Log compatibility version specifier – differentiates version 10.x logs from release 11.x logs by showing the format of log records in a database, database dump, or transaction log dump. This constant is stored in the database and dump headers and is used by Adaptive Server to detect the format of log records during recovery.

Cache bindings and loading databases

If you dump a database and load it onto a server with different cache bindings, you should be aware of cache bindings for a database and the objects in the database. You may want to load the database onto a different server for tuning or development work, or you may need to load a database that you dropped from a server whose cache bindings have changed since you made the dump.

When you bring a database online after recovery or by using online database after a load, Adaptive Server verifies all cache bindings for the database and database objects. If a cache does not exist, Adaptive Server writes a warning to the error log, and the binding in sysattributes is marked as invalid. Here is an example of the message from the error log:

```
Cache binding for database ’5’, object ’208003772’,
index ’3’ is being marked invalid in Sysattributes.
```

Invalid cache bindings are not deleted. If you create a cache of the same name and restart Adaptive Server, the binding is marked as valid and the cache is used. If you do not create a cache with the same name, you can bind the object to another cache or allow it to use the default cache.

In the following sections, which discuss cache binding topics, *destination server* refers to the server where the database is being loaded, and *original server* refers to the server where the dump was made.

If possible, re-create caches that have the same names on the destination server as the bindings on the original server. You may want to configure pools in exactly the same manner if you are using the destination database for similar purposes or for performance testing and development that may be ported back to the original server. If you are using the destination database for decision support or for running dbcc commands, you may want to configure pools to allow more space in 16K memory pools.

Databases and cache bindings

Binding information for databases is stored in master..sysattributes. No information about database binding is stored in the database itself. If you use load database to load the dump over an existing database that is bound to a cache, and you do not drop the database before you issue the load command, this does not affect the binding.

If the database that you are loading was bound to a cache on the original server, you can:

- Bind the database on the destination server to a cache configured for the needs on that server, or
- Configure pools in the default data cache on the destination server for the needs of the application there, and do not bind the database to a named data cache.

Database objects and cache bindings

Binding information for objects is stored in the sysattributes table in the database itself. If you frequently load the database onto the destination server, the simplest solution is to configure caches of the same name on the destination server.

If the destination server is not configured with caches of the same name as the original server, bind the objects to the appropriate caches on the destination server after you bring the database online, or be sure that the default cache is configured for your needs on that server.

Checking on cache bindings

Use sp_helpcache to display the cache bindings for database objects, even if the cache bindings are invalid.

The following SQL statements reproduce cache binding commands from the information in a user database's sysattributes table:

```
/* create a bindcache statement for tables */

select "sp_bindcache "+ char_value + ", "
   + db_name() + ", " + object_name(object)
from sysattributes
where class = 3
   and object_type = "T"
```

```
/* create a bindcache statement for indexes */

select "sp_bindcache "+ char_value + ", "
   + db_name() + ", "   + i.name
from sysattributes, sysindexes i
where class = 3
   and object_type = "I"
   and i.indid = convert(tinyint, object_info1)
   and i.id = object
```

Cross-database constraints and loading databases

If you use the references constraint of create table or alter database to reference tables across databases, you may encounter problems when you try to load a dump of one of these databases.

- If tables in a database reference a dumped database, referential integrity errors result if you load the database with a different name or on a different server from where it was dumped. To change the name or location of a database when you reload it, use alter table in the referencing database to drop all external referential integrity restraints before you dump the database.
- Loading a dump of a referenced database that is earlier than the referencing database could cause consistency issues or data corruption. As a precaution, each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump both affected databases.
- Dump all databases that reference each other at the same time. To guard against synchronization problems, put both databases in single-user mode for the dumps. When loading the databases, bring both databases online at the same time.

Cross-database constraints can become inconsistent if you:

- Do not load database dumps in chronological order (for example, you load a dump created on August 12, 1997, after one created on August 13), or
- Load a dump into a database with a new name.

If you do not load, cross-database constraints can become inconsistent.

To remedy this problem:

1. Put both databases in single-user mode.
2. Drop the inconsistent referential constraint.
3. Check the data consistency with a query such as:

```
select foreign_key_col from table1
where foreign_key not in
(select primary_key_col from otherdb..othertable)
```

4. Fix any data inconsistency problems.
5. Re-create the constraint.

CHAPTER 28 Restoring the System Databases

This chapter explains how to restore the master, model, and sybsystemprocs databases.

Topics covered in this chapter include:

Topic	Page
What does recovering a system database entail?	889
Symptoms of a damaged master database	890
Recovering the master database	890
Recovering the model database	904
Recovering the sybsystemprocs database	905
Restoring system tables with disk reinit and disk refit	908

What does recovering a system database entail?

The recovery procedure for system databases depends on the database involved and the problems that you have on your system. In general, recovery may include:

- Using load database to load backups of these databases,
- Using dataserver, installmaster, and installmodel to restore the initial state of these databases, or
- A combination of the above tasks.

To make the recovery of system databases as efficient as possible:

- Do not store user databases or any databases other than master, tempdb, and model on the master device.
- Always keep up-to-date printouts of important system tables.
- Always back up the master database after performing actions such as initializing database devices, creating or altering databases, or adding new server logins.

Symptoms of a damaged *master* database

A damaged master database can be caused by a media failure in the area on which master is stored or by internal corruption in the database. You'll know if your master database is damaged if:

- Adaptive Server cannot start.
- There are frequent or debilitating segmentation faults or input/output errors.
- dbcc reports damage during a regular check of your databases.

Recovering the *master* database

This section describes how to recover the master database and to rebuild the master device. It assumes:

- The master database is corrupt, or the master device is damaged.
- You have up-to-date printouts of the system tables, listed in "Backing up master and keeping copies of system tables" on page 24.
- The master device contains *only* the master database, tempdb, and model.
- You have an up-to-date backup of the master database, and you have not initialized any devices or created or altered any databases since last dumping master.
- Your server uses the default sort order.

You can also use these procedures to move your master database to a larger master device.

The *Troubleshooting Guide* provides more complete coverage of recovery scenarios.

About the recovery process

Special procedures are needed because of the central, controlling nature of the master database and the master device. Tables in master configure and control all Adaptive Server's functions, databases, and data devices. The recovery process:

- Rebuilds the master device to its default state when you first installed a server
- Restores the master database to the default state
- Restores the master database to its condition at the time of your last backup

During the early stages of recovering the master database, you cannot use the system stored procedures.

Summary of recovery procedure

You must follow the steps below to restore a damaged master device. *Each step is discussed in more detail on the following pages.*

Step	See
Find hard copies of the system tables needed to restore disks, databases and logins.	"Step one: find copies of system tables" on page 892
Shut down Adaptive Server, and use dataserver to build a new master database and master device.	"Step two: build a new master device" on page 893
Restart Adaptive Server in master-recover mode.	"Step three: start Adaptive Server in master-recover mode" on page 894
Re-create the master database's allocations in sysusages exactly.	"Step four: re-create device allocations for master" on page 895
Update Backup Server's network name in the sysservers table.	"Step five: check your Backup Server sysservers information" on page 899
Verify that your Backup Server is running.	"Step six: verify that your Backup Server ss running" on page 900
Use load database to load the most recent database dump of master. Adaptive Server stops automatically after successfully loading master.	"Step seven: load a backup of master" on page 900

Step	See
Update the number of devices configuration parameter in the configuration file.	"Step eight: update the number of devices configuration parameter" on page 901.
Restart Adaptive Server in single-user mode.	"Step nine: restart Adaptive Server in master-recover mode" on page 901
Verify that the backup of master has the latest system tables information.	"Step ten: check system tables to verify current backup of master" on page 901
Restart Adaptive Server.	"Step eleven: restart Adaptive Server" on page 902
Check syslogins if you have added new logins since the last backup of master.	"Step twelve: restore server user IDs" on page 902
Restore the model database.	"Step thirteen: restore the model database" on page 903
Compare hard copies of sysusages and sysdatabases with the new online version, run dbcc checkalloc on each database, and examine the important tables in each database.	"Step fourteen: check Adaptive Server" on page 903
Dump the master database.	"Step fifteen: back up master" on page 904

Step one: find copies of system tables

Find copies of the system tables that you have saved to a file: sysdatabases, sysdevices, sysusages, sysloginroles, and syslogins. You can use these to guarantee that your system has been fully restored at the completion of this process.

For information on preparing for disaster recovery by making copies of the system tables to a file, see "Backing up master and keeping copies of system tables" on page 24.

Step two: build a new master device

Before you run dataserver, check your most recent copy of sysusages. If it has only one line for dbid 1, your master database has only one disk allocation piece, and you can go to “Step five: check your Backup Server sysservers information” on page 899.

Shut down Adaptive Server, if it is running, and rebuild the master device. When rebuilding the master device, you must specify the device size.

Before you begin, it is important that you remember to:

- Use a new device, preserving the old device in case you encounter problems. The old device may provide crucial information.
- Shut down Adaptive Server before you use any dataserver command. If you use dataserver on a master device that is in use by Adaptive Server, the recovery procedure will fail when you attempt to load the most recent backup of master.

Run dataserver (UNIX) or sqlsrvr (Windows NT) to build a new master device and to install a copy of a “generic” master database. Give the full name and full size for your master device.

Note You must give dataserver a size as large as or larger than the size originally used to configure Adaptive Server. If the size is too small, you will get error messages when you try to load your databases.

The following example rebuilds a 17MB master device.

On UNIX platforms:

```
dataserver -d /dev/rsd1f -b17M
```

On Windows NT:

```
sqlsrvr -d d:\devices\master.dat -b17M
```

After you run dataserver, the password for the default “sa” account reverts to NULL.

For details on the dataserver utility, see the *Utility Guide*.

Step three: start Adaptive Server in master-recover mode

Start Adaptive Server in master-recover mode with the -m (UNIX and Windows NT) option.

On UNIX platforms, make a copy of the runserver file, naming it *m_RUN_server_name*. Edit the new file, adding the parameter -m to the dataserver command line. Then start the server in master-recover mode:

```
    startserver -f m_RUN_server_name
```

On Windows NT, start Adaptive Server from the command line using the sqlsrver command. Specify the -m parameter in addition to other necessary parameters. For example:

```
sqlsrver.exe -dD:\Sybase\DATA\MASTER.dat -sPIANO -eD:\Sybase\install\errorlog
-iD:\Sybase\ini -MD:\Sybase -m
```

See the *Utility Guide* for the complete syntax of these commands.

When you start Adaptive Server in master-recover mode, only one login of one user—the System Administrator—is allowed. Immediately following a dataserver command on the master database, only the “sa” account exists, and its password is NULL.

Warning! Some sites have automatic jobs that log in to the server at start-up with the “sa” login. Be sure these are disabled.

Master-recover mode is necessary because the generic master database created with dataserver does not match the actual situation in Adaptive Server. For example, the database does not know about any of your database devices. Any operations on the master database could make recovery impossible or at least much more complicated and time-consuming.

An Adaptive Server started in master-recover mode is automatically configured to allow direct updates to the system tables. Certain other operations (for example, the checkpoint process) are disallowed.

Warning! Ad hoc changes to system tables are dangerous—some changes can render Adaptive Server unable to run. Make only the changes described in this chapter, and always make the changes in a user-defined transaction.

Step four: re-create device allocations for *master*

If more than one row for dbid 1 appears in your hard copy of sysusages, you need to increase the size of master so that you can load the dump. You must duplicate the vstart value for each allocation for master in sysusages. This is easiest to do if you have a copy of sysusages ordered by vstart.

In the simplest cases, additional allocations to master require only the use of alter database. In more complicated situations, you must allocate space for other databases to reconstruct the exact vstart values needed to recover master.

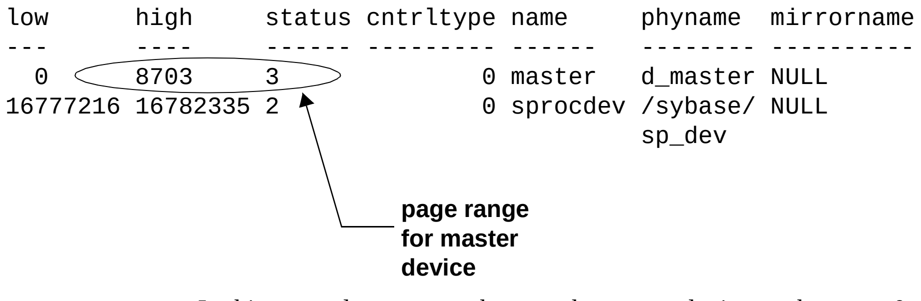
In addition to the master database, tempdb (dbid = 2) and model (dbid = 3) are located wholly or partially on the master device.

Determining which allocations are on the master device

Note The examples in this section assume a server that uses 2K logical pages.

To determine which vstart values represent allocations on the master device, look at the sysdevices table. It shows the low and high values for each device. Databases devices always have a cntrltype of 0; the following example does not include the rows for tape devices.

Figure 28-1: Determining allocations on the master device

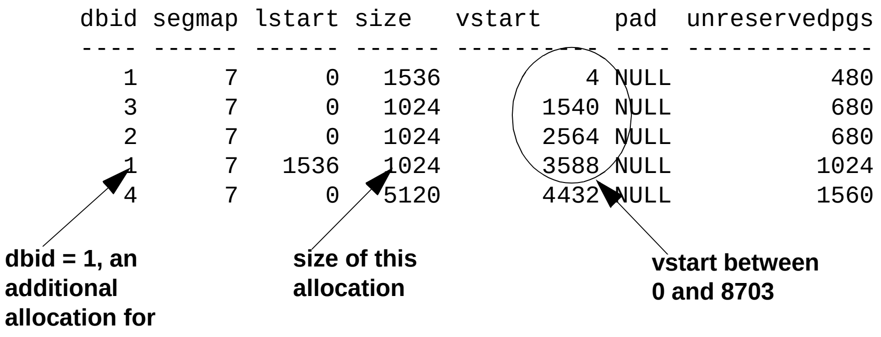


```
low      high     status cntrltype name     phyname  mirrorname
---      ----     ------ --------- ------   -------- ----------
  0      8703     3              0 master   d_master NULL
16777216 16782335 2              0 sprocdev /sybase/ NULL
                                            sp_dev
```

In this example, page numbers on the master device are between 0 and 8703, so any database allocation with sysusages.vstart values in this range represent allocations on the master device.

Check all rows for master (except the first) in your saved sysusages output. Here is sample sysusages information, ordered by vstart:

Figure 28-2: Sample output from sysusages



```
dbid segmap lstart size   vstart     pad  unreservedpgs
---- ------ ------ ------ ---------- ---- -------------
   1      7      0   1536          4 NULL           480
   3      7      0   1024       1540 NULL           680
   2      7      0   1024       2564 NULL           680
   1      7   1536   1024       3588 NULL          1024
   4      7      0   5120       4432 NULL          1560
```

dbid = 1, an additional allocation for
size of this allocation
vstart between 0 and 8703

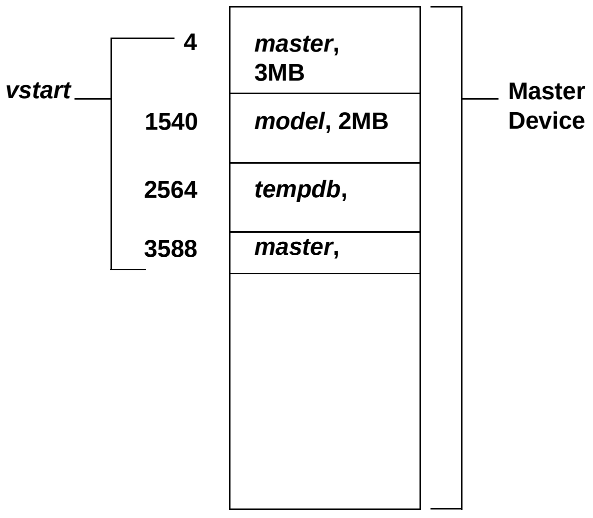
In this example, the first four rows have vstart values between 4 and 3588. Only dbid 4 is on another device.

dataserver re-creates the first three rows, so sysusages in your newly rebuilt master database should match your hard copy.

The fourth row shows an additional allocation for master with vstart = 3588 and *size* = 1024.

Figure 28-3 shows the storage allocations for the above sysusages data.

Figure 28-3: Allocations on a master device



In Figure 28-3, you need only to issue an alter database command to increase the size of the master database. To determine the size to provide for this command, look at the *size* column for the second allocation to master. Divide by 512. In this example, the additional row for master indicates an allocation of 1024 data pages, so the correct parameter is 2, the result of 1024/512.

Use that result for the alter database command. Log in to the server as "sa." Remember that dataserver has set the password for this account to NULL. Issue the alter database command. For the example above, use:

```
alter database master on master = "2M"
```

Check the *size* and vstart values for the new row in sysusages.

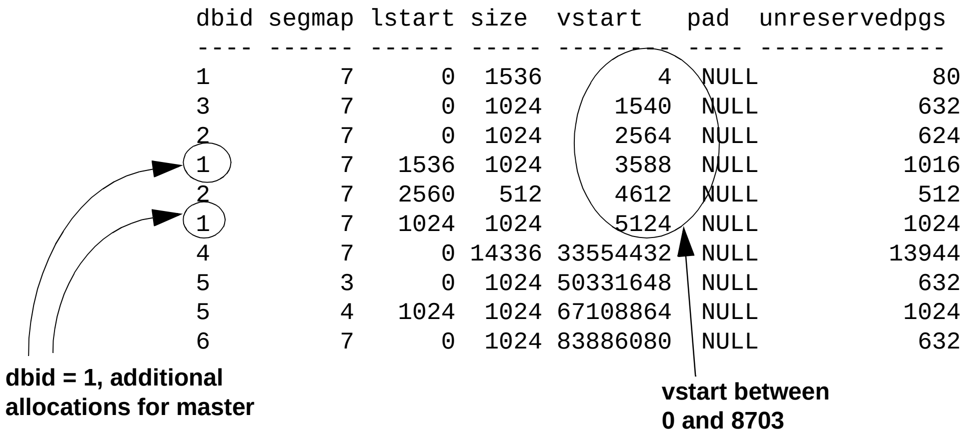
Creating additional allocations

Your output from sysusages may have more allocations on the master device if:

- You have upgraded Adaptive Server from an earlier version
- A System Administrator has increased the size of master, model, or tempdb on the master device

You must restore these allocations up to the last row for the master database, dbid 1. Here is an example of sysusages showing additional allocations on the master device, in vstart order:

Figure 28-4: Sample sysusages output with additional allocations



```
dbid segmap lstart size  vstart   pad  unreservedpgs
---- ------ ------ ----- -------- ---- -------------
1         7      0  1536        4 NULL            80
3         7      0  1024     1540 NULL           632
2         7      0  1024     2564 NULL           624
1         7   1536  1024     3588 NULL          1016
2         7   2560   512     4612 NULL           512
1         7   1024  1024     5124 NULL          1024
4         7      0 14336 33554432 NULL         13944
5         3      0  1024 50331648 NULL           632
5         4   1024  1024 67108864 NULL          1024
6         7      0  1024 83886080 NULL           632
```

dbid = 1, additional allocations for master
vstart between 0 and 8703

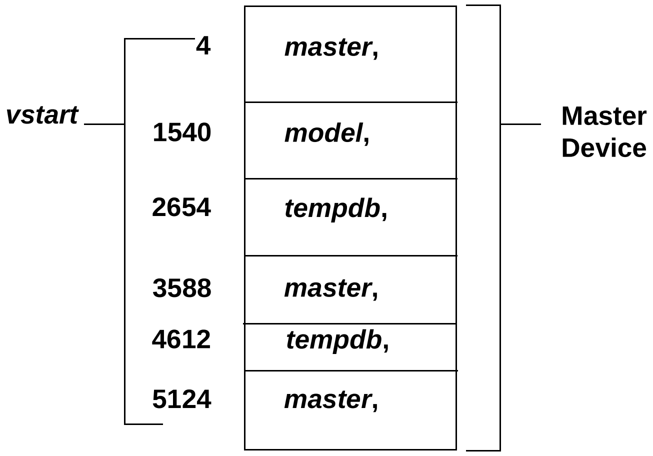
This copy of sysusages shows the following allocations on the master device (excluding the three created by dataserver):

- One for master, dbid = 1, *size* = 1024, vstart = 3588
- One for tempdb, dbid = 2, *size* = 512, vstart = 4612
- Another allocation for master, dbid = 1, *size* = 1024, vstart = 5124

The final allocations in this output are not on the master device.

Figure 28-5 shows the allocations on the master device.

Figure 28-5: Complex allocations on a master device



You need to issue a set of alter database and create database commands to re-create all the allocations with the correct sizes and vstart values. If sysusages lists additional allocations on the master device after the last allocation for master, you do not have to re-create them.

To determine the size for the create database and alter database commands, divide the value shown in the *size* column of the sysusages output by 512.

To reconstruct the allocation, issue these commands, in this order:

- To restore the first allocation to master, dbid 1, *size* = 1024:

```
alter database master on default = “2M”
```

- To allocate more space to tempdb, dbid 2, *size* = 512:

```
    alter database tempdb on default = “1M”
```

- To add the final allocation to master, dbid 1, *size* = 1024:

```
    alter database master on default = “1M”
```

You need to restore only the allocations up to and including the last line for the master database. When you load the backup of master, this table is completely restored from the dump.

At this point, carefully check the current sysusages values with the values in your hard copy:

- If all of the vstart and *size* values for master match, go to “Step five: check your Backup Server sysservers information” on page 899.
- If the values do not match, an attempt to load the master database will almost certainly fail. Shut down the server, and begin again by running dataserver. See “Step two: build a new master device” on page 893.
- If your sysusages values look correct, go to “Step five: check your Backup Server sysservers information” on page 899.

Step five: check your Backup Server *sysservers* information

Log in to the server as “sa,” using a null password.

If the network name of your Backup Server is not SYB_BACKUP, you must update sysservers so that Adaptive Server can communicate with its Backup Server. Check the Backup Server name in your interfaces file, and issue this command:

```
select *
from sysservers
where srvname = "SYB_BACKUP"
```

Check the srvnetname in the output from this command. If it matches the interfaces file entry for the Backup Server for your server, go to “Step six: verify that your Backup Server ss running” on page 900.

If the reported srvnetname is *not* the same as the Backup Server in the interfaces file, you must update sysservers. The example below changes the Backup Server’s network name to PRODUCTION_BSRV:

```
begin transaction
update sysservers
set srvnetname = "PRODUCTION_BSRV"
```

```
where srvname = "SYB_BACKUP"
```

Execute this command, and check to be sure that it modified only one row. Issue the select command again, and verify that the correct row was modified and that it contains the correct value. If update modified more than one row, or if it modified the wrong row, issue a rollback transaction command, and attempt the update again.

If the command correctly modified the Backup Server's row, issue a commit transaction command.

Step six: verify that your Backup Server ss running

On UNIX platforms, use the showserver command to verify that your Backup Server is running; restart your Backup Server if necessary. See showserver and startserver in the *Utility Guide*.

On Windows NT, a locally installed Sybase Central and the Services Manager show whether Backup Server is running.

See the *Utility Guide* for the commands to start Backup Server.

Step seven: load a backup of *master*

Load the most recent backup of the master database. Here are examples of the load commands:

On UNIX platforms:

```
load database master from "/dev/nrmt4"
```

On Windows NT:

```
load database master from "\\.\TAPE0"
```

See Chapter 27, "Backing Up and Restoring User Databases," for information on command syntax.

After load database completes successfully, Adaptive Server shuts down. Watch for any error messages during the load and shut down processes.

Step eight: update the *number of devices* configuration parameter

Perform this step only if you use more than the default number of database devices. Otherwise, go to “Step nine: restart Adaptive Server in master-recover mode” on page 901.

Configuration values are not available to Adaptive Server until after recovery of the master database, so you need to instruct Adaptive Server to read the appropriate value for the number of devices parameter from a configuration file at start-up.

If your most recent configuration file is not available, edit a configuration file to reflect the correct value for the number of devices parameter.

Edit the runserver file. Add the -c parameter to the end of the dataserver or sqlsrver command, specifying the name and location of the configuration file. When Adaptive Server starts, it reads the parameter values from the specified configuration file.

Step nine: restart Adaptive Server in master-recover mode

Use startserver to restart Adaptive Server in master-recover mode (see “Step three: start Adaptive Server in master-recover mode” on page 894). Watch for error messages during recovery.

Loading the backup of master restores the “sa” account to its previous state. It restores the password on the “sa” account, if one exists. If you used sp_locklogin to lock this account before the backup was made, the “sa” account will now be locked. Perform the rest of the recovery steps using an account with the System Administrator role.

Step ten: check system tables to verify current backup of *master*

If you have backed up the master database since issuing the most recent disk init, create database, or alter database command, then the contents of sysusages, sysdatabases, and sysdevices will match your hard copy.

Check the sysusages, sysdatabases, and sysdevices tables in your recovered server against your hard copy. Look especially for these problems:

- If any devices in your hard copy are not included in the restored sysdevices, then you have added devices since your last backup, and you must run disk reinit and disk refit. For information on using these commands, see "Restoring system tables with disk reinit and disk refit" on page 908.
- If any databases listed in your hard copy are not listed in your restored sysdatabases table, you have added a database since the last time you backed up master. You must run disk refit (see "Restoring system tables with disk reinit and disk refit" on page 908).

Step eleven: restart Adaptive Server

Restart Adaptive Server in normal (multiuser) mode.

Step twelve: restore server user IDs

Check your hard copy of syslogins and your restored syslogins table. Look especially for the following situations and reissue the appropriate commands, as necessary:

- If you have added server logins since the last backup of master, reissue the sp_addlogin commands.
- If you have dropped server logins, reissue the sp_droplogin commands.
- If you have locked server accounts, reissue the sp_locklogin commands.
- Check for other differences caused by the use of sp_modifylogin by users or by System Administrators.

Make sure that the suids assigned to users are correct. Mismatched suid values in databases can lead to permission problems, and users may not be able to access tables or run commands.

An effective technique for checking existing suid values is to perform a union on each sysusers table in your user databases. You can include master in this procedure, if users have permission to use master.

For example:

```
select suid, name from master..sysusers
```

```
union
select suid, name from sales..sysusers
union
select suid, name from parts..sysusers
union
select suid, name from accounting..sysusers
```

If your resulting list shows skipped suid values in the range where you need to redo the logins, you must add placeholders for the skipped values and then drop them with sp_droplogin or lock them with sp_locklogin.

Step thirteen: restore the *model* database

Restore the model database:

- Load your backup of model, if you keep a backup.
- If you do not have a backup:
  - Run the installmodel script:

    On most platforms:

```
cd $SYBASE/scripts
isql -Usa -Ppassword -Sserver_name < installmodel
```

On Windows NT:

```
cd $SYBASE/scripts
isql -Usa -Ppassword -Sserver_name < instmodl
```

  - Redo any changes you made to model.

Step fourteen: check Adaptive Server

Check Adaptive Server carefully:

1. Compare your hard copy of sysusages with the new online version.
2. Compare your hard copy of sysdatabases with the new online version.
3. Run dbcc checkalloc on each database.

4 Examine the important tables in each database.

Warning! If you find discrepancies in sysusages, call Sybase Technical Support.

Step fifteen: back up *master*

When you have completely restored the master database and have run full dbcc integrity checks, back up the database using your usual dump commands.

Recovering the *model* database

This section describes recovery of the model database when only the model database needed to be restored. It includes instructions for these scenarios:

- You have not made any changes to model, so you need to restore only the generic model database.
- You have changed model, and you have a backup.
- You have changed model, and you do not have a backup.

Restoring the generic *model* database

dataserver can restore the model database without affecting master.

Warning! Shut down Adaptive Server before you use any dataserver command.

On UNIX platforms:

```
dataserver -d /devname  -x
```

On Windows NT:

```
sqlsrvr -d physicalname -x
```

Restoring *model* from a backup

If you can issue the model successfully, you can restore your model database from a backup with load database.

If you cannot use the database:

1. Follow the instructions for "Restoring the generic model database" on page 904.
2. If you have changed the size of model, reissue alter database.
3. Load the backup with load database.

Restoring *model* with no backup

If you have changed your model database, and you do not have a backup:

- Follow the steps for "Restoring the generic model database" on page 904.
- Reissue all the commands you issued to change model.

Recovering the *sybsystemprocs* database

The sybsystemprocs database stores the system procedures that are used to modify and report on system tables. If your routine dbcc checks report damage, and you do not keep a backup of this database, you can restore it using installmaster. If you do keep backups of sybsystemprocs, you can restore it with load database.

Restoring *sybsystemprocs* with *installmaster*

1. Check to see what logical device currently stores the database. If you can still use sp_helpdb, issue:

```
                       sp_helpdb sybsystemprocs
name                  db_size       owner            dbid
        created
        status
------------------- ------------- ---------------- ------
```

```
sybsystemprocs           28.0 MB sa                    4
        Aug 07, 1993
        trunc log on chkpt

device_fragments   size        usage            free kbytes
------------------ ----------- ---------------- -----------
sprocdev           28.0 MB     data and log            3120
```

The "device_fragments" column indicates that the database is stored on sprocdev.

If you cannot use sp_helpdb, this query reports the devices used by the database and the amount of space on each device:

```
select sysdevices.name, sysusages.size / 512
from sysdevices, sysdatabases, sysusages
where sysdatabases.name = "sybsystemprocs"
  and sysdatabases.dbid = sysusages.dbid
  and sysdevices.low <= sysusages.size + vstart
  and sysdevices.high >= sysusages.size + vstart -1
name
---------------- -------
sprocdev              28
```

2 Drop the database:

```
drop database sybsystemprocs
```

If the physical disk is damaged, use dbcc dbrepair to drop the database and then use sp_dropdevice to drop the device. If necessary, use disk init to initialize a new database device. See Chapter 16, "Initializing Database Devices," for more information on disk init.

3 Re-create the sybsystemprocs database on the device, using the size returned by the query under step 1:

```
create database sybsystemprocs
```

```
        on sprocdev = 28
```

Note The required size for sybsystemprocs may be different for your operating system. See the installation documentation for your platform for the correct size.

4 Run the installmaster script.

Warning! Running *installmaster* repeatedly can change the distribution of index values in such a way that the sysprocedures table will require much more disk space to store the same amount of data. To avoid this problem, drop and re-create sybsystemprocs before running *installmaster*.

On UNIX platforms:

```
cd $SYBASE/scripts
isql -Usa -Ppassword -Sserver_name <
installmaster
```

On Windows NT:

```
cd $SYBASE/scripts
isql -Usa -Ppassword -Sserver_name < instmstr
```

5 If you have made any changes to permissions in sybsystemprocs, or if you have added your own procedures to the database, you must redo the changes.

Restoring *sybsystemprocs* with *load database*

If you write system procedures and store them in sybsystemprocs, there are two ways to recover them if the database is damaged:

- Restore the database from installmaster, as described in step 4 under “Restoring sybsystemprocs with installmaster” on page 905. Then re-create the procedures by reissuing the create procedure commands.
- Keep backups of the database, and load them with load database.

If you choose to keep a backup of the database, be sure that the complete backup fits on one tape volume or that more than one Adaptive Server is able to communicate with your Backup Server. If a dump spans more than one tape volume, issue the change-of-volume command using sp_volchanged, which is stored in sybsystemprocs. You cannot issue that command in the middle of recovering a database.

Following are sample load commands:

On UNIX:

```
load database sybsystemprocs from "/dev/nrmt4"
```

On Windows NT:

```
load database sybsystemprocs from "\\.\TAPE0"
```

Restoring system tables with *disk reinit* and *disk refit*

When you are restoring the master database from a dump that does not reflect the most recent disk init or create database and alter database commands, follow the procedures in this section to restore the proper information in the sysusages, sysdatabases, and sysdevices tables.

Restoring *sysdevices* with *disk reinit*

If you have added any database devices since the last dump—that is, if you have issued a disk init command—you must add each new device to sysdevices with disk reinit. If you saved scripts from your original disk init commands, use them to determine the parameters for disk reinit (including the original value of vstart). If the size you provide is too small, or if you use a different vstart value, you may corrupt your database.

If you did not save your disk init scripts, look at your most recent hard copy of sysdevices to determine some of the correct parameters for disk reinit. You will still need to know the value of vstart if you used a custom vstart in the original disk init command.

Table 28-1 describes the disk reinit parameters and their corresponding sysdevices data:

Table 28-1: Using sysdevices to determine disk reinit parameters

***disk reinit* parameter**	***sysdevices* data**	**Notes**
name	name	Use the same name, especially if you have any scripts that create or alter databases or add segments.
physname	*phyname*	Must be full path to device.
vdevno	*low*/16777216	Not necessary to use the same value for *vdevno*, but be sure to use a value not already in use.
size	(*high* -*low*) +1	Extremely important to provide correct size information.

You can also obtain information on devices by reading the error log for *name*, *physname*, and *vdevno*, and using operating system commands to determine the size of the devices.

If you store your sybsystemprocs database on a separate physical device, be sure to include a disk reinit command for sybsystemprocs, if it is not listed in sysdevices.

After running disk reinit, compare your sysdevices table to the copy you made before running dataserver.

disk reinit can be run only from the master database and only by a System Administrator. Permission cannot be transferred to other users. Its syntax is:

```
disk reinit
    name = "device_name",
    physname = "physical_name",
    [vdevno = virtual_device_number,]
    size = number_of_blocks
     [, vstart = virtual_address,
    cntrltype = controller_number]
```

For more information on disk reinit, see the discussion of disk init in Chapter 16, “Initializing Database Devices,” or the *Adaptive Server Reference Manual*.

Restoring *sysusages* and *sysdatabase* with *disk refit*

If you have added database devices or created or altered databases since the last database dump, use disk refit to rebuild the sysusages and sysdatabases tables.

disk refit can be run only from the master database and only by a System Administrator. Permission cannot be transferred to other users. Its syntax is:

```
disk refit
```

Adaptive Server shuts down after disk refit rebuilds the system tables. Examine the output while disk refit runs and during the shutdown process to determine whether any errors occurred.

Warning! Providing inaccurate information in the disk reinit command may lead to permanent corruption when you update your data. Be sure to check Adaptive Server with dbcc after running disk refit.

CHAPTER 29

Managing Free Space with Thresholds

When you create or alter a database, you allocate a finite amount of space for its data and log segments. As you create objects and insert data, the amount of free space in the database decreases.

This chapter explains how to use thresholds to monitor the amount of free space in a database segment.

Topics include:

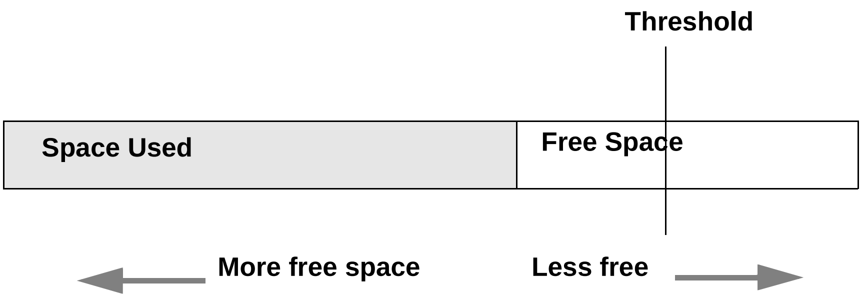
Monitoring free space with the last-chance threshold

All databases have a **last-chance threshold**, including master. The threshold is an estimate of the number of free log pages that are required to back up the transaction log. As you allocate more space to the log segment, Adaptive Server automatically adjusts the last-chance threshold.

When the amount of free space in the log segment falls below the last-chance threshold, Adaptive Server automatically executes a special stored procedure called sp_thresholdaction. (You can specify a different last-chance threshold procedure with sp_modifythreshold.)

Figure 29-1 illustrates a log segment with a last-chance threshold. The shaded area represents log space that has already been used; the unshaded area represents free log space. The last-chance threshold has not yet been crossed.

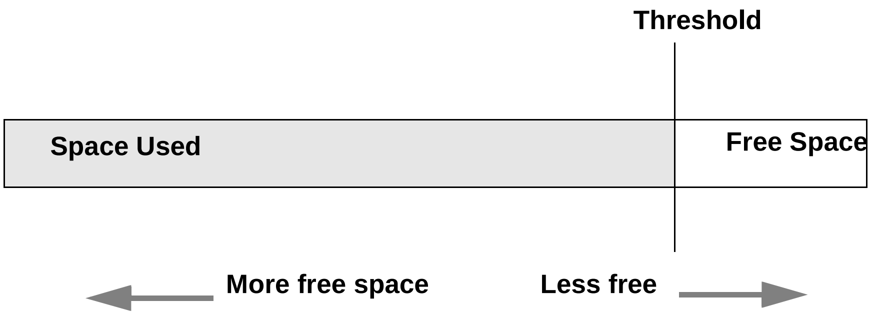
Figure 29-1: Log segment with a last-chance threshold



Crossing the threshold

As users execute transactions, the amount of free log space decreases. When the amount of free space crosses the last-chance threshold, Adaptive Server executes sp_thresholdaction:

Figure 29-2: Executing sp_thresholdaction when the last-chance threshold is reached



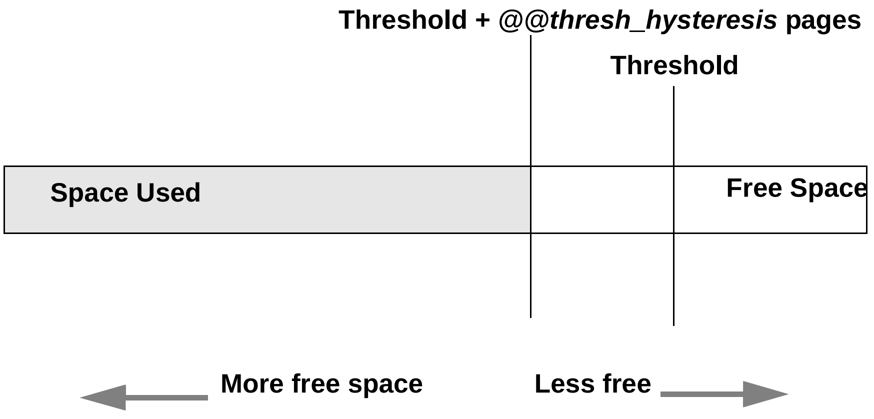
Controlling how often *sp_thresholdaction* executes

Adaptive Server uses a *hysteresis value*, the global variable *@@thresh_hysteresis*, to control how sensitive thresholds are to variations in free space.

A threshold is deactivated after it executes its procedure, and remains inactive until the amount of free space in the segment rises *@@thresh_hysteresis* pages above the threshold. This prevents thresholds from executing their procedures repeatedly in response to minor fluctuations in free space. You cannot change the value of *@@thresh_hysteresis*.

For example, when the threshold in Figure 29-2 executes sp_thresholdaction, it is deactivated. In Figure 29-3, the threshold is reactivated when the amount of free space increases by the value of *@@thresh_hysteresis*:

Figure 29-3: Free space must rise by @@thresh_hysteresis to reactivate threshold



Rollback records and the last-chance threshold

Adaptive Server version 11.9 and later include **rollback records** in the transaction logs. Rollback records are logged whenever a transaction is rolled back. Servers save enough space to log a rollback record for every update belonging to an open transaction. If a transaction completes successfully, no rollback records are logged, and the space reserved for them is released.

In long-running transactions, rollback records can reserve large amounts of space. However, because Adaptive Server does not allocate the space reserved for the rollback records to the transaction log, this space is not reported by either sp_spaceused or dbcc checktable.

If the last chance threshold for the transaction log fires when it seems to have sufficient space, it may be the space reserved for rollbacks that is causing the problem. See “Determining the current space for rollback records” on page 915 for more information.

Calculating the space for rollback records

To calculate the increased amount of space to add to a transaction log to accommodate rollback records, estimate:

- The number of update records in the transaction log that are likely to belong to already rolled-back transactions
- The maximum number of update records in the transaction log that are likely to belong to open transactions at any one time

Update records are the records that change the timestamp value. They include changes to datapages, indexpages, allocation pages, and so on.

Each rollback record requires approximately 60 bytes of space, or 3 one hundredths of a page. Thus, the calculation for including rollback records (RRs) in the transaction log is:

Added space, in pages = (logged RRs + # open updates) X 3/100

You may also want to add log space to compensate for the effects of rollback records on the last-chance threshold and on user-defined thresholds, as described in the following sections.

Using lct_admin to determine the free log space

Use the logsegment_freepages to determine the amount of free space your dedicated log segment has. The syntax is:

lct_admin("logsegment_freepages", *database_id*)

for example, to see the amount of free pages for the pubs2 database log segment:

```
select lct_admnin(“logsegment_freepages”, 4


-------------------
                 79
```

Determining the current space for rollback records

To determine the number of pages a database currently reserved for rollbacks, issue lct_admin with the reserved_for_rollbacks parameter. The partial syntax for lct_admin is:

select lct_admin ("reserved_for_rollbacks", *dbid*, 0)

The number of pages returned are the number reserved, but not yet allocated, for rollback records.

For example, to determine the number of pages reserved for rollbacks in the pubs2 database (which has a pubid of 5), issue the following:

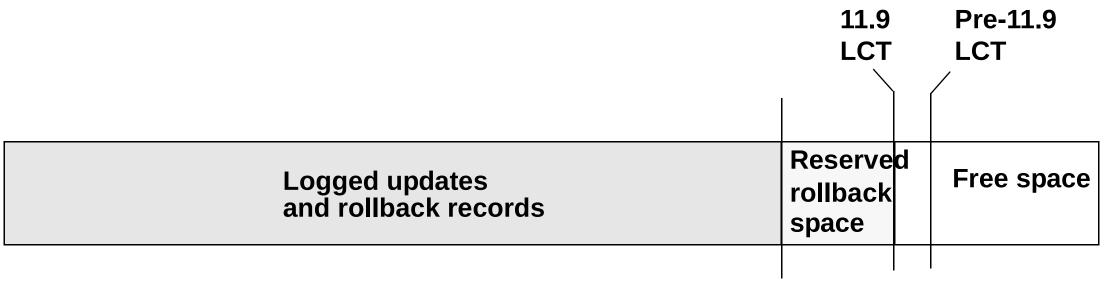
```
select lct_admin("reserved_for_rollbacks", 5, 0)
```

See the Adaptive Server Reference Manual for more information about lct_admin

Effect of rollback records on the last-chance threshold

Adaptive Servers that use rollback records must reserve additional room for the last-chance threshold. The last-chance threshold (“LCT”) is also likely to be reached sooner because of the space used by already logged rollback records and the space reserved against open transactions for potential rollback records. Figure 29-4 illustrates how space is used in a transaction log with rollback records:

Figure 29-4: Space used in log with rollback records



In Figure 29-4, in the 11.9 Adaptive Server, log space is occupied by logged rollback records for closed transactions that did not complete successfully. In addition, space is reserved for rollback records that may need to be logged, if any of the currently open transactions do not complete successfully. Together, the space for logged rollback records and for potential rollback records is likely to be considerably greater than the extra space for the last-chance threshold. Consequently, transaction logs that use rollback records reach the last-chance threshold significantly sooner than transaction logs that do not use rollback records, even if the size of the transaction log is not increased.

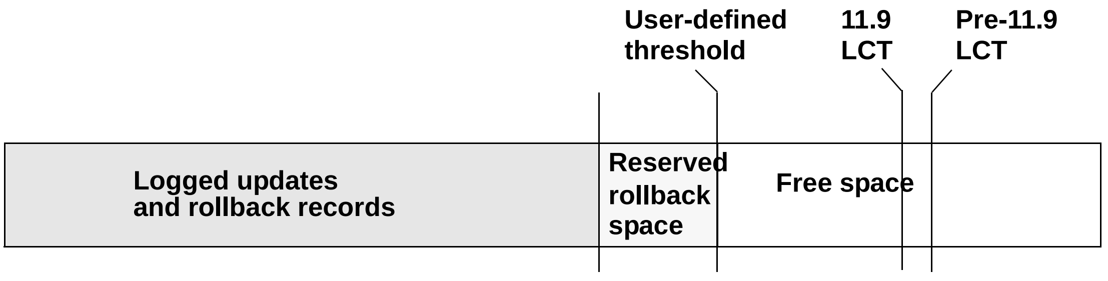
In general, about 18 percent more log space is reserved for the last-chance threshold in 11.9 and later Adaptive Servers than in earlier versions. For example, for a transaction log of 5000 pages, version 11.5 reserves 264 pages and version 11.9 reserves 312 pages. This is an increase of 18.18 percent between version 11.5 and 11.9.

User-defined thresholds

Because rollback records occupy extra space in the transaction log, there is less free space after the user-defined threshold for completing a dump than in versions of Adaptive Server that do not use rollback records (See Figure 29-4). However, the loss of space for a dump because of the increased last-chance threshold is likely to be more than compensated for by the space reserved for rollback records for open transactions.

Upgrading to version that uses rollback records affects user-defined thresholds similarly to the way it effects last-chance thresholds. Figure 29-5 illustrates the effect of upgrading to an Adaptive Server using rollback records on a user-defined threshold:

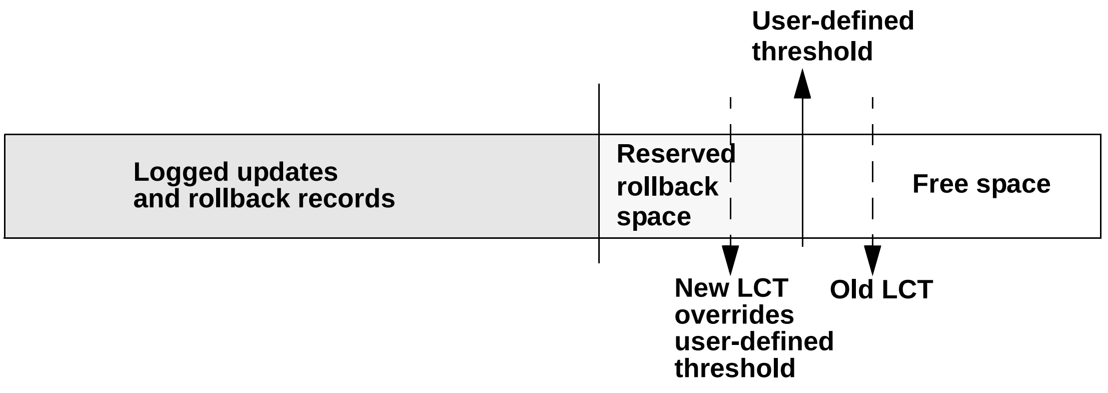
Figure 29-5: Effect of upgrading on user-defined thresholds



A user-defined threshold such as the one in Figure 29-5 is often used to initiate a dump transaction. The threshold is set so there is enough room to complete the dump before the last-chance threshold is reached and all open transactions in the log are suspended.

In databases that use mixed log and data, the last-chance threshold moves dynamically, and its value can be automatically configured to be less than the user-defined threshold. If this happens, the user-defined threshold is disabled, and the last chance threshold fires before the user-defined threshold is reached, as shown in Figure 29-6:

Figure 29-6: LCT firing before user-defined threshold


User-defined threshold
Logged updates and rollback records
Reserved rollback space
Free space
New LCT overrides user-defined threshold
Old LCT

The user-defined threshold is re-enabled if the value of last-chance threshold is configured to be greater than the user-defined threshold (for example, if the last chance threshold is reconfigured for the value of “Old LCT” in Figure 29-6).

In databases with a separate log segment, the log has a dedicated amount of space and the last-chance threshold is static. The user-defined threshold is not affected by the last-chance threshold.

Last-chance threshold and user log caches for shared log and data segments

Every database in an Adaptive Server has a last-chance threshold and all databases allow transactions to be buffered in a user log cache. When you initially create a database with shared log and data segments, its last-chance threshold is based on the size of the model database. As soon as data is added and logging activity begins, the last-chance threshold is recalculated dynamically, based on available space and currently open transactions. The last-chance threshold of a database with separate log and data segments is based on the size of the log segment and does not vary dynamically.

To get the current last-chance threshold of any database, you can use lct_admin with the reserve parameter and a specification of 0 log pages:

```
select lct_admin("reserve",0)
```

The last-chance threshold for a database is stored in the systhresholds table and is also accessible through sp_helpthreshold. However, note that:

- sp_helpthreshold returns user-defined thresholds and other data, as well as an up-to-date value for the last-chance threshold. Using lct_admin is simpler if you need only the current last-chance threshold. Either of these values produce the most current value for the last-chance threshold.
- For a database with shared log and data segments, the last-chance threshold value in systhresholds may *not* be the current last-chance threshold value.

Reaching last-chance threshold suspends transactions

The default behavior of Adaptive Server is to suspend open transactions until additional log space is created. Transactions suspended because of the last-chance threshold can be terminated using the abort parameter of the lct_admin system function, described in "Using lct_admin abort to abort suspended transactions," below. For information about configuring Adaptive Server to automatically abort suspended processes, see "Automatically aborting or suspending processes" on page 921.

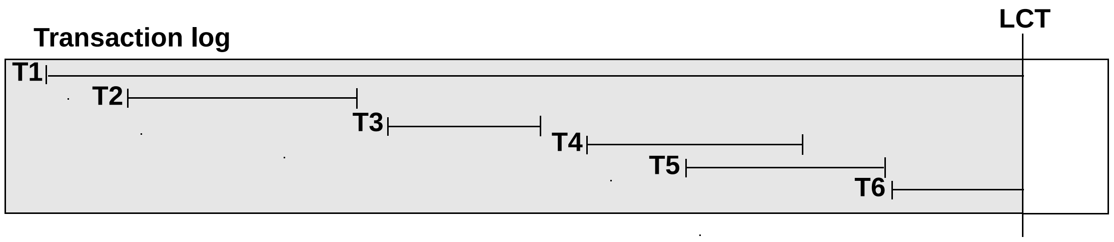
Using *lct_admin abort* to abort suspended transactions

When the transaction log reaches the last-chance threshold, all becomes made available. Typically, space is created by dumping the transaction log, since this removes committed transactions from the beginning of the log. However, if one or more transactions at the beginning of the log is still open, it prevents a dump of the transaction log.

Use lct_admin abort to terminate suspended transactions that are preventing a transaction log dump. Since terminating a transaction

closes it, this allows the dump to proceed. Figure 29-7 illustrates a possible scenario for using lct_admin abort:

Figure 29-7: Example of when to use of lct_admin abort



In Figure 29-7, a transaction log has reached its LCT, and open transactions T1 and T6 are suspended. Because T1 is at the beginning of the log, it prevents a dump from removing closed transactions T3 through T5 and creating space for continued logging. Terminating T1 with lct_admin abort allows you to close T1 so that a dump can clear transactions T1 through T5 from the log.

lct_admin abort replaces lct_admin unsuspend.

lct_admin abort **syntax**

The syntax for lct_admin abort is:

lct_admin("abort", {*process_id [*, *database_id]}*)

Before you can abort a transaction, you must first determine its ID. See “Getting the process ID for the oldest open transaction,” below for information about determining the transaction’s pid.

To terminate the oldest transaction, enter the process ID (spid) of the process that initiated the transaction. This also terminates any other suspended transactions in the log that belong to the specified process.

For example, if process 83 holds the oldest open transaction in a suspended log, and you want to terminate the transaction, enter:

```
select lct_admin("abort", 83)
```

This also terminates any other open transactions belonging to process 83 in the same transaction log.

To terminate all open transactions in the log, enter:

```
select lct_admin("abort", 0, 12)
```

Getting the process ID for the oldest open transaction

Use the following query to find the spid of the oldest open transaction in a transaction log that has reached its last-chance threshold:

```
use master
go
select dbid, spid from syslogshold
where dbid = db_id("name_of_database")
```

For example, to find the oldest running transaction on the pubs2 database:

```
select dbid, spid from syslogshold
where dbid = db_id ("pubs2")
dbid   spid
------ ------
     7      1
```

Using *alter database* when the master database reaches the last-chance threshold

When the last-chance threshold on the master database is reached, you can use alter database to add space to the master database's transaction log. This allows more activity in the server by causing suspended transactions in the log to become active. However, while the master transaction log is at its last-chance threshold, you cannot use alter database to make changes in other databases. Thus, if both master and another database reach their last-chance thresholds, you would first need to use alter database to add log space to the master database, and then use it again to add log space to the second database.

Automatically aborting or suspending processes

By design, the last-chance threshold allows enough free log space to record a dump transaction command. There may not be enough room to record additional user transactions against the database.

When the last-chance threshold is crossed, Adaptive Server suspends user processes and displays the message:

```
Space available in the log segment has fallen
critically low in database 'mydb'. All future
modifications to this database will be suspended
until the log is successfully dumped and space
becomes available.
```

Only commands that are not recorded in the transaction log (select or readtext) and commands that might be necessary to free additional log space (dump transaction, dump database, and alter database) can be executed.

Using *abort tran on log full* to abort transactions

To configure the last-chance threshold to automatically abort open transactions, rather than suspend them:

sp_dboption *database_name* "abort tran on log full", true

If you upgrade from a previous version of Adaptive Server, the newly upgraded server retains the abort tran on log full setting.

Waking suspended processes

After dump transaction frees sufficient log space, suspended processes automatically awaken and complete. If writetext or select into has resulted in unlogged changes to the database since the last backup, the last-chance threshold procedure cannot execute dump transaction. When this occurs, make a copy of the database with dump database, then truncate the log with dump transaction.

If this does not free enough space to awaken the suspended processes, you may need to increase the size of the transaction log. Use the log on option of alter database to allocate additional log space.

As a last resort, System Administrators can use the sp_who command to determine which processes are in a log suspend status and then use the kill command to kill the sleeping process.

Adding, changing, and deleting thresholds

The Database Owner or System Administrator can create additional thresholds to monitor free space on any segment in the database. Additional thresholds are called **free-space thresholds**. Each database can have up to 256 thresholds, including the last-chance threshold.

sp_addthreshold, sp_modifythreshold, and sp_dropthreshold allow you to create, change, and delete thresholds. To prevent users from accidentally affecting thresholds in the wrong database, these procedures require that you specify the name of the current database.

Displaying information about existing thresholds

Use sp_helpthreshold to get information about all thresholds in a database. Use sp_helpthreshold *segment_name* to get information about the thresholds on a particular segment.

The following example displays information about the thresholds on the database's default segment. Since "default" is a reserved word, you must enclose it in quotation marks. The output of sp_helpthreshold shows that there is one threshold on this segment set at 200 pages. The 0 in the "last chance" column indicates that this is a free-space threshold instead of a last-chance threshold:

```
                                     sp_helpthreshold "default"
segment name   free pages   last chance?   threshold procedure
------------   ----------   ------------   -------------------
default        200                     0   space_dataseg

(1 row affected, return status = 0)
```

Thresholds and system tables

The system table systhresholds holds information about thresholds. sp_helpthreshold uses this table to provide its information. In addition to information about segment name, free page, last-chance status, and the name of the threshold procedure, the table also records the server user ID of the user who created the threshold and the roles had at the moment the threshold was created.

Adaptive Server gets information about how much free space is remaining in a segment—and whether to activate a threshold—from the built-in system function curunreservedpgs().

Adding a free-space threshold

Use sp_addthreshold to create free-space thresholds. Its syntax is:

sp_addthreshold *dbname*, *segname*, *free_space*, *proc_name*

The *dbname* must specify the name of the current database. The remaining parameters specify the segment whose free space is being monitored, the size of the threshold in database pages, and the name of a stored procedure.

When the amount of free space on the segment falls below the threshold, an internal Adaptive Server process executes the associated procedure. This process has the permissions of the user who created the threshold when he or she executed sp_addthreshold, less any permissions that have since been revoked.

Thresholds can execute a procedure in the same database, in another user database, in sybsystemprocs, or in master. They can also call a remote procedure on an Open Server. sp_addthreshold does not verify that the threshold procedure exists when you create the threshold.

Changing a free-space threshold

Use sp_modifythreshold to associate a free-space threshold with a new threshold procedure, free-space value, or segment. sp_modifythreshold drops the existing threshold and creates a new one in its place. Its syntax is:

> sp_modifythreshold *dbname* , *segname* , *free_space*
> [, *new_proc_name* [, *new_free_space*
> [, *new_segname*]]]

where*dbname* is the name of the current database, and *segname* and *free_space* identify the threshold that you want to change.

For example, to execute a threshold procedure when free space on the segment falls below 175 pages rather than below 200 pages, enter:

```
sp_modifythreshold mydb, "default", 200, NULL, 175
```

In this example, NULL acts as a placeholder so that *new_free_space* falls in the correct place in the parameter list. The name of the threshold procedure is not changed.

The person who modifies the threshold becomes the new threshold owner. When the amount of free space on the segment falls below the threshold, Adaptive Server executes the threshold procedure with the owner's permissions at the time he or she executed sp_modifythreshold, less any permissions that have since been revoked.

Specifying a new last-chance threshold procedure

You can use sp_modifythreshold to change the name of the procedure associated with the last-chance threshold. You *cannot* use it to change the amount of free space or the segment name for the last-chance threshold.

sp_modifythreshold requires that you specify the number of free pages associated with the last-chance threshold. Use sp_helpthreshold to determine this value.

The following example displays information about the last-chance threshold, and then specifies a new procedure, sp_new_thresh_proc, to execute when the threshold is crossed:

```
sp_helpthreshold logsegment
```

```
segment name   free pages   last chance?   threshold procedure
------------   ----------   ------------   -------------------
logsegment     40                      1   sp_thresholdaction

(1 row affected, return status = 0)
```

```
sp_modifythreshold mydb, logsegment, 40,
sp_new_thresh_proc
```

Dropping a threshold

Use sp_dropthreshold to remove a free-space threshold from a segment. Its syntax is:

sp_dropthreshold *dbame*, *segname*, *free_space*

The *dbname* must specify the name of the current database. You must specify both the segment name and the number of free pages, since there can be several thresholds on a particular segment. For example:

```
sp_dropthreshold mydb, "default", 200
```

Creating a free-space threshold for the log segment

When the last-chance threshold is crossed, all transactions are aborted or suspended until sufficient log space is freed. In a production environment, this can have a heavy impact on users. Adding a correctly placed free-space threshold on your log segment can minimize the chances of crossing the last-chance threshold.

The additional threshold should dump the transaction log often enough that the last-chance threshold is rarely crossed. It should not dump it so often that restoring the database requires the loading of too many tapes.

This section helps you determine the best place for a second log threshold. It starts by adding a threshold with a *free_space* value set at 45 percent of log size and adjusts this threshold based on space usage at your site.

Adding a log threshold at 45 percent of log size

Use the following procedure to add a log threshold with a *free_space* value set at 45 percent of log size.

1 Determine the log size in pages:

```
select sum(size)
from master..sysusages
where dbid = db_id("database_name")
and (segmap & 4) = 4
```

2 Use sp_addthreshold to add a new threshold with a *free_space* value set at 45 percent. For example, if the log's capacity is 2048 pages, add a threshold with a *free_space* value of 922 pages:

```
sp_addthreshold mydb, logsegment, 922,
thresh_proc
```

3 Create a simple threshold procedure that dumps the transaction log to the appropriate devices. For more information about creating threshold procedures, see "Creating threshold procedures" on page 930.

Testing and adjusting the new threshold

Use dump transaction to make sure your transaction log is less than 55 percent full. Then use the following procedure to test the new threshold:

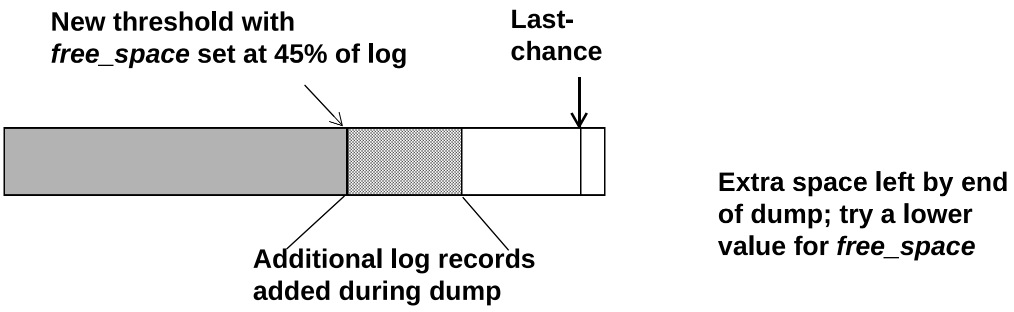
1 Fill the transaction log by simulating routine user action. Use automated scripts that perform typical transactions at the projected rate.

   When the 45 percent free-space threshold is crossed, your threshold procedure will dump the transaction log. Since this is not a last-chance threshold, transactions will not be suspended or aborted; the log will continue to grow during the dump.

2 While the dump is in progress, use sp_helpsegment to monitor space usage on the log segment. Record the maximum size of the transaction log just before the dump completes.

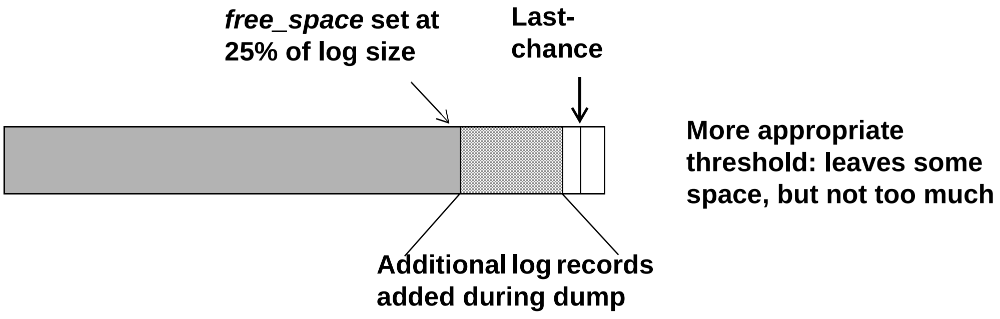
3 If considerable space was left in the log when the dump completed, you may not need to dump the transaction log so soon, as shown in Figure 29-8:

Figure 29-8: Transaction log with additional threshold at 45 percent


New threshold with
free_space set at 45% of log
Last-
chance
Extra space left by end of dump; try a lower value for *free_space*
Additional log records
added during dump

Try waiting until only 25 percent of log space remains, as shown in Figure 29-9:

Figure 29-9: Moving threshold leaves less free space after dump

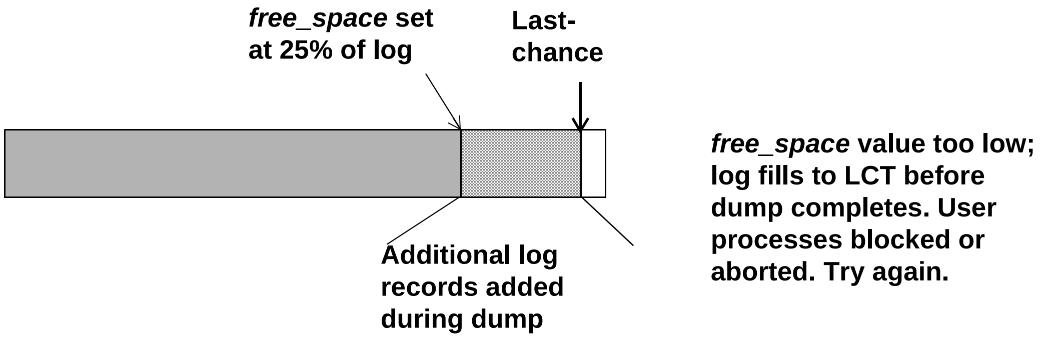


Use sp_modifythreshold to adjust the *free_space* value to 25 percent of the log size. For example:

```
sp_modifythreshold mydb, logsegment, 512,
thresh_proc
```

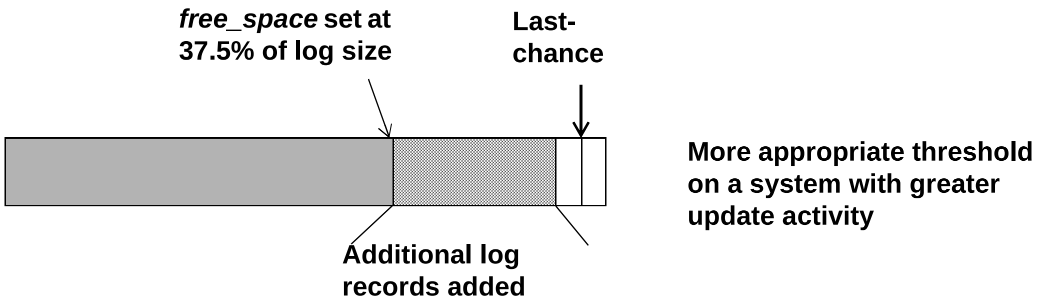
4 Dump the transaction log and test the new *free_space* value. If the last-chance threshold is crossed before the dump completes, you are not beginning the dump transaction soon enough, as shown in Figure 29-10:

Figure 29-10: Additional log threshold does not begin dump early enough


free_space set at 25% of log
Last-chance
free_space value too low; log fills to LCT before dump completes. User processes blocked or aborted. Try again.
Additional log records added during dump

25 percent free space is not enough. Try initiating the dump transaction when the log has 37.5 percent free space, as shown in Figure 29-11:

Figure 29-11: Moving threshold leaves enough free space to complete dump


free_space set at 37.5% of log size
Last-chance
More appropriate threshold on a system with greater update activity
Additional log records added

Use sp_modifythreshold to change the *free_space* value to 37.5 percent of log capacity. For example:

```
sp_modifythreshold mydb, logsegment, 768,
    thresh_proc
```

Creating additional thresholds on other segments

You can create free-space thresholds on data segments as well as on log segments. For example, you might create a free-space threshold on the default segment used to store tables and indexes. You would also create an associated stored procedure to print messages in your error log when space on the default segment falls below this threshold. If you monitor the error log for these messages, you can add space to the database device before your users encounter problems.

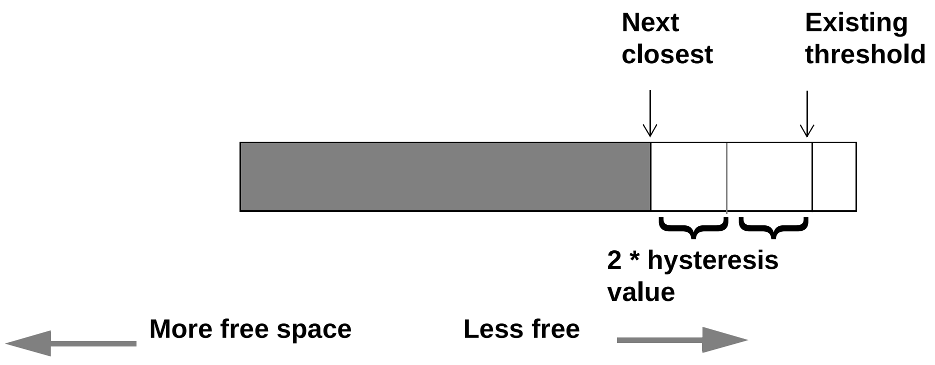
The following example creates a free-space threshold on the default segment of mydb. When the free space on this segment falls below 200 pages, Adaptive Server executes the procedure space_dataseg:

```
sp_addthreshold mydb, "default", 200, space_dataseg
```

Determining threshold placement

Each new threshold must be at least twice the *@@thresh_hysteresis* value from the next closest threshold, as shown in Figure 29-12:

Figure 29-12: Determining where to place a threshold



To see the hysteresis value for a database, use:

```
select @@thresh_hysteresis
```

In this example, a segment has a threshold set at 100 pages, and the hysteresis value for the database is 64 pages. The next threshold must be at least 100 + (2 * 64), or 228 pages.

```
select @@thresh_hysteresis
-----------
        64
sp_addthreshold mydb, user_log_dev, 228,
    sp_thresholdaction
```

Creating threshold procedures

Sybase does not supply threshold procedures. You must create these procedures yourself to ensure that they are tailored to your site's needs.

Suggested actions for a threshold procedure include writing to the server's error log and dumping the transaction log to increase the amount of log space. You can also execute remote procedure calls to an Open Server or to XP Server. For example, if you include the following command in sp_thresholdaction, it executes the procedure mail_me on an Open Server:

```
exec openserv...mail_me @dbname, @segment
```

See Chapter 15, "Using Extended Stored Procedures," in the *Transact-SQL User's Guide* for more information on using extended stored procedures and XP Server.

This section provides some guidelines for writing threshold procedures, as well as two sample procedures.

Declaring procedure parameters

Adaptive Server passes four parameters to a threshold procedure:

- *@dbname*, varchar(30), which contains the database name
- *@segmentname*, varchar(30), which contains the segment name
- *@space_left*, int, which contains the space-left value for the threshold

- *@status*, int, which has a value of 1 for last-chance thresholds and 0 for other thresholds

These parameters are passed by position rather than by name. Your procedure can use other names for these parameters, but must declare them in the order shown and with the datatypes shown.

Generating error log messages

You should include a print statement near the beginning of your procedure to record the database name, segment name, and threshold size in the error log. If your procedure does not contain a print or raiserror statement, the error log will not contain any record of the threshold event.

The process that executes threshold procedures is an internal Adaptive Server process. It does not have an associated user terminal or network connection. If you test your threshold procedures by executing them directly (that is, using execute *procedure_name*) during a terminal session, you see the output from the print and raiserror messages on your screen. When the same procedures are executed by reaching a threshold, the messages go to the error log. The messages in the log include the date and time.

For example, if sp_thresholdaction includes this statement:

```
print "LOG DUMP: log for ’%1!’ dumped", @dbname
```

Adaptive Server writes this message to the error log:

```
00: 92/09/04 15:44:23.04 server: background task message: LOG DUMP: log for
’pubs2’ dumped
```

Dumping the transaction log

If your sp_thresholdaction procedure includes a dump transaction command, Adaptive Server dumps the log to the devices named in the procedure. dump transaction truncates the transaction log by removing all pages from the beginning of the log, up to the page just before the page that contains an uncommitted transaction record.

When there is enough log space, suspended transactions are awakened. If you abort transactions rather than suspending them, users must resubmit them.

Generally, dumping to a disk is *not* recommended, especially to a disk that is on the same machine or the same disk controller as the database disk. However, since threshold-initiated dumps can take place at any time, you may want to dump to disk and then copy the resulting files to offline media. (You will have to copy the files back to the disk to reload them.)

Your choice will depend on:

- Whether you have a dedicated dump device online, loaded and ready to receive dumped data
- Whether you have operators available to mount tape volumes during the times when your database is available
- The size of your transaction log
- Your transaction rate
- Your regular schedule for dumping databases and transaction logs
- Available disk space
- Other site-specific dump resources and constraints

A simple threshold procedure

Following is a simple procedure that dumps the transaction log and prints a message to the error log. Because this procedure uses a variable (*@dbname*) for the database name, it can be used for all databases in Adaptive Server:

```
create procedure sp_thresholdaction
    @dbname varchar(30),
    @segmentname varchar(30),
    @free_space int,
    @status int
as
dump transaction @dbname
    to tapedump1
print "LOG DUMP: ’%1!’ for ’%2!’ dumped",
        @segmentname, @dbname
```

A more complex procedure

The following threshold procedure performs different actions, depending on the value of the parameters passed to it. Its conditional logic allows it to be used with both log and data segments.

The procedure:

- Prints a "LOG FULL" message if the procedure was called as the result of reaching the log's last-chance threshold. The status bit is 1 for the last-chance threshold and 0 for all other thresholds. The test if (@status&1) = 1 returns a value of "true" only for the last-chance threshold.
- Verifies that the segment name provided is the log segment. The segment ID for the log segment is always 2, even if the name has been changed.
- Prints "before" and "after" size information on the transaction log. If the log did not shrink significantly, a long-running transaction may be causing the log to fill.
- Prints the time the transaction log dump started and stopped, helping gather data about dump durations.
- Prints a message in the error log if the threshold is not on the log segment. The message gives the database name, the segment name and the threshold size, letting you know that the data segment of a database is filling up.

```
create procedure sp_thresholdaction
     @dbname           varchar(30),
     @segmentname      varchar(30),
     @space_left       int,
     @status           int
as
declare @devname varchar(100),
           @before_size int,
           @after_size int,
           @before_time datetime,
           @after_time datetime,
           @error int

/*
** if this is a last-chance threshold, print a LOG FULL msg
** @status is 1 for last-chance thresholds,0 for all others
*/
if (@status&1) = 1
```

```
begin
      print "LOG FULL: database ’%1!’", @dbname
end

/*
** if the segment is the logsegment, dump the log
** log segment is always "2" in syssegments
*/
if @segmentname = (select name from syssegments
               where segment = 2)
begin
     /* get the time and log size
     ** just before the dump starts
     */
     select  @before_time = getdate(),
        @before_size = reserved_pgs(id, doampg)
      from sysindexes
      where sysindexes.name = "syslogs"

      print "LOG DUMP: database ’%1!’, threshold ’%2!’",
        @dbname, @space_left

      select @devname = "/backup/" + @dbname + "_" +
        convert(char(8), getdate(),4) + "_" +
        convert(char(8), getdate(), 8)

      dump transaction @dbname to @devname
      /* error checking */
      select @error = @@error
      if @error != 0
      begin
           print "LOG DUMP ERROR: %1!", @error
      end

      /* get size of log and time after dump */
      select @after_time = getdate(),
           @after_size = reserved_pgs(id, doampg)
           from sysindexes
           where sysindexes.name = "syslogs"

      /* print messages to error log */
      print "LOG DUMPED TO: device ’%1!", @devname
      print "LOG DUMP PAGES: Before: ’%1!’, After ’%2!’",
           @before_size, @after_size
      print "LOG DUMP TIME: %1!, %2!", @before_time, @after_time
end        /* end of ’if segment = 2’ section */
```

```
else       /* this is a data segment, print a message */
begin
      print "THRESHOLD WARNING: database ’%1!’, segment ’%2!’ at ’%3!’
pages", @dbname, @segmentname, @space_left
end
```

Deciding where to put a threshold procedure

Although you can create a separate procedure to dump the transaction log for each threshold, it is easier to create a single threshold procedure that is executed by all log segment thresholds. When the amount of free space on a segment falls below a threshold, Adaptive Server reads the systhresholds table in the affected database for the name of the associated stored procedure, which can be any of:

- A remote procedure call to an Open Server
- A procedure name qualified by a database name (for example, sybsystemprocs.dbo.sp_thresholdaction)
- An unqualified procedure name

If the procedure name does not include a database qualifier, Adaptive Server looks in the database where the shortage of space occurred. If it cannot find the procedure there, and if the procedure name begins with the characters “sp_”, Adaptive Server looks for the procedure in the sybsystemprocs database and then in master database.

If Adaptive Server cannot find the threshold procedure, or cannot execute it, it prints a message in the error log.

Disabling free-space accounting for data segments

Use the no free space acctg option to sp_dboption, followed by the checkpoint command, to disable free-space accounting on non-log segments. You *cannot* disable free-space accounting on the log segment.

When you disable free-space accounting, only the thresholds on your log segment monitor space usage; threshold procedures on your data segments will not execute when these holds are crossed. Disabling free-space accounting speeds recovery time because free-space counts are not recomputed during recovery for any segment except the log segment.

The following example turns off free-space accounting for the production database:

```
sp_dboption production,
     "no free space acctg", true
```

Warning! If you disable free-space accounting, system procedures cannot provide accurate information about space allocation.

Index