CHAPTER 20

Backup and Data Recovery

About this chapter

This chapter explains how to use transaction log files to protect your data, how to back up your database files and the log file, and the recovery procedures for system and media failures.

Contents

System and media failures

Adaptive Server Anywhere has features to protect your data from two categories of computer failure: **system failure** and **media failure**.

- **System Failure** A system failure occurs when the computer or operating system goes down while there are partially completed transactions. This could occur when the computer is inappropriately turned off or rebooted, when another application causes the operating system to crash, or because of a power failure.
- **Media Failure** A media failure occurs when the database file, the file system, or the device storing the database file becomes unusable.

Recovery from failure

When failures occur, the recovery mechanism treats transactions properly, as **atomic** units of work: any incomplete transaction is rolled back and any committed transaction is preserved in the transaction log. This ensures that even in the event of failure, the data in your database remains in a consistent state.

Make regular backups

You should make regular backups of your database files so that you can recover your database in the case of a media failure. The transaction log (which you should store on a separate device from the database for greater security) is used to recover information put into the database since the last full backup.

Database logs

The following logs protect your data from system and media failure.

- The checkpoint log.
- The rollback log.
- The transaction log.

All of these logs play a role in data recovery. Each database running on a server has its own checkpoint log, rollback log, and transaction log. Optionally, you can also maintain a mirror of the transaction log for greater protection of vital data.

The checkpoint log

Checkpoint log purpose

The checkpoint log is used during database recovery after a system failure or improper shutdown of the server. The checkpoint log is stored in the database file.

A database file is composed of pages. Before a page is updated (made **dirty**), a copy of the original is always made. The copied pages are the **checkpoint** log.

Dirty pages are not written immediately to the database file on disk. For improved performance, they are cached in memory and only written to the database file when the cache is full or the server has no pending requests. A **checkpoint** is a point at which all dirty pages are written to disk. Following a checkpoint, the checkpoint log is deleted.

Reasons for a checkpoint

A checkpoint can occur for several reasons:

- The server is shut down
- The amount of time since the last checkpoint exceeds the database option CHECKPOINT_TIME
- The estimated time to do a recovery operation exceeds the database option RECOVERY_TIME
- The server is idle long enough to write all dirty pages
- A transaction issues a CHECKPOINT statement
- The server is running without a transaction log, and a transaction is committed

Checkpoint priority

The priority of writing dirty pages to the disk increases as the time and the amount of work since the last checkpoint grows. The priority is determined by the following factors:

- **Checkpoint Urgency** The time that has elapsed since the last checkpoint, as a percentage of the checkpoint time setting of the database. The server `-gc` command line option controls the maximum desired time, in minutes, between checkpoints. You can also set the desired time using the CHECKPOINT_TIME option.
- **Recovery Urgency** A heuristic to estimate the amount of time required to recover the database if it fails right now. The server `-gu` command line option controls the maximum desired time, in minutes, for recovery in the event of system failure. You can also set the desired time using the RECOVERY_TIME option.

The checkpoint and recovery urgencies are important only if the server does not have enough idle time to write dirty pages.

For a description of the command-line options, see "The database server" on page 12 of the book *Adaptive Server Anywhere Reference Manual*.

How the database decides when to checkpoint

Optional information
You do not need to know the information in this section for most purposes. It is provided as background information for those who wish to understand more about how the server works.

The idle I/O task

The writing of dirty pages to disk is carried out by a task within the server called the **idle I/O task**. This task shares processing time with other database tasks, according to a priority. The lower the priority of the idle I/O task, the less time it gets.

There is a threshold for the number of dirty pages, below which writing of database pages does not take place.

When the database is busy, the urgency is low, and the cache only has a few dirty pages, the idle I/O task runs at a very low priority and no writing of dirty pages takes place.

Once the urgency exceeds 30%, the priority of the idle I/O task is increased. At intervals, the priority is increased again. As the urgency becomes high, the engine shifts its primary focus to writing dirty pages until the number gets below the threshold again. However, the engine only writes out pages during the idle I/O task if the number of dirty pages is greater than the threshold.

If, because of other activity in the database, the number of dirty pages falls to zero, and if the urgency is 50% or more, then a checkpoint takes place automatically, since it is a convenient time.

Both the checkpoint urgency and recovery urgency values increase in value until the checkpoint occurs, at which point they drop to zero. They do not decrease otherwise.

The rollback log

As changes are made to the contents of tables, a **rollback log** is kept for the purpose of canceling changes. It is used to process the ROLLBACK statement for recovering from system failure. There is a separate rollback log for each connection. When a transaction is complete, the rollback log contents for that connection are deleted. The rollback logs are stored in the database file.

The transaction log

All changes to the database are stored in the transaction log in the order that they occur. Inserts, updates, deletes, commits, rollbacks, and database schema changes are all logged. The transaction log is also called a **forward log file**.

The transaction log is stored in a separate file.

What you should do

The transaction log is optional. If you run with no transaction log, a checkpoint is carried out whenever a transaction is committed. The checkpoint ensures that all committed transactions are written to the disk. Checkpoints can be time consuming, so you should run with a transaction log for improved performance as well as protection against media failure and corrupted databases.

For greater protection, you can maintain two identical transaction logs in tandem. This is called transaction log mirroring.

☞ For information on creating a database with a mirrored transaction log, see "The Initialization utility" on page 84 of the book *Adaptive Server Anywhere Reference Manual*. For information on changing an existing database to use a mirrored transaction log, see "The Transaction Log utility" on page 105 of the book *Adaptive Server Anywhere Reference Manual*.

Keep the transaction log on a separate device

The transaction log is not kept in the main database file. The filename of the transaction log can be set when the database is initialized, or at any other time when the server is not running, using the Log utility.

To protect against media failure, the transaction log should be written to a different device than the database file. Some machines with two or more hard drives have only one physical disk drive with several logical drives or partitions. If you want protection against media failure, make sure that you have a machine with two storage devices, or use a storage device on a network file server.

By default, the transaction log is put on the same device and in the same directory as the database—this does not protect against media failure.

Primary key definitions affect transaction log size

Updates and deletes on tables that do not have a primary key or unique index cause the entire contents of the affected rows to be entered in the transaction log. If a primary key is defined, the engine needs only to record the primary key column values to uniquely identify a row. If the table contains many columns or wide columns, the transaction log pages will fill up much faster (reducing performance) if no primary key is defined. And if *dbtran* is used on the transaction log, it produces a very large command file.

This affects updates and deletes, but not inserts, which must always log all column values.

If a primary key does not exist, the engine looks for a UNIQUE NOT NULL index on the table (or a UNIQUE constraint). A UNIQUE index that allows null values is not sufficient.

Performance tip
Placing the transaction log on a separate device can also result in improved performance by eliminating the need for disk head movement between the transaction log and the main database file.

Converting transaction logs to SQL

The transaction log is not human-readable. The Log Translation utility (*dbtran*) can be used to convert a transaction log into a SQL command file, which can serve as an audit trail of changes made to the database. The following command uses *dbtran* to convert a transaction log:

```
dbtran sample.log changes.sql
```

You can also convert a transaction log to a SQL command file from Sybase Central.

For more information on the log translation utility, see "The Log Translation utility" on page 98 of the book *Adaptive Server Anywhere Reference Manual*.

Recovering uncommitted database changes

The transaction log contains a record of everything, including transactions that were never committed. By converting the transaction log to a SQL command file and choosing the option to include uncommitted transactions (for example, by running *dbtran* with the `-a` switch) you can recover transactions that were accidentally canceled by a user. If the `-a` option is not chosen, the log translation utility omits transactions that were rolled back. While this is not a common procedure, it can prove useful for exceptional cases.

Using a transaction log mirror

A transaction log mirror is an identical copy of the transaction log, maintained at the same time as the transaction log. Transaction log mirrors are used to allow complete recovery in the case of media failure on the log device.

Every time a database change is written to the transaction log, it is also written to the transaction log mirror file. By default, a mirrored transaction log is not used, but you can choose to use one when creating a database or you can make an existing database use a mirrored transaction log.

Why use a transaction log mirror?

A mirrored transaction log provides extra protection of critical data. For example, at a consolidated database in a SQL Remote setup, replication relies on the transaction log, and if the transaction log is damaged or becomes corrupt, data replication can fail.

A mirrored transaction log carries out automatic validation of the transaction log on database startup.

There is a performance penalty for using a mirrored log, as each database log write operation must be carried out twice. The performance penalty depends on the nature and volume of database traffic and on the physical configuration of the database and logs.

If you are using a mirrored transaction log, and get an error while trying to write to one of them (for example, if the disk is full), the server or server stops. The purpose of a transaction log mirror is to ensure complete recoverability in the case of media failure on either log device; this purpose would be lost if the server continued with a single log.

Where to store the transaction log mirror

A transaction log mirror should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.

Creating and dropping a transaction log mirror

Transaction log mirrors can be created at the following times:

- When you create a database, using the Initialization utility.
- At any other time that the database is not running, using the Transaction Log utility (*dblog*).
- A mirror for a write file transaction log can be created along with the write file using the write file utility (mirroring can be added later, using the transaction log utility).

Notes

- You cannot choose to use a transaction log mirror without using a transaction log.
- The default file extension for transaction log mirrors is .MLG.

Creating a database with a transaction log mirror

You can choose to maintain a transaction log mirror when you create a database. This option is available either from the CREATE DATABASE statement, from Sybase Central or from the *dbinit* command-line utility.

- From Sybase Central, the transaction log mirror option is part of the Create Database utility.

  ☞ For more information, see the Sybase Central online Help.

- The following command line (which should be entered on one line) initializes a database named *company.db*, with a transaction log kept on a different device and a mirror on a third device.

  ```
  dbinit -t d:\log_dir\company.log -m
  e:\mirr_dir\company.mlg c:\db_dir\company.db
  ```

  By default, a transaction log is used but no transaction log mirror is created.

☞ For a full description of initialization options, see "Initialization utility options" on page 85 of the book *Adaptive Server Anywhere Reference Manual*.

Starting a transaction log mirror for an existing database

You can choose to maintain a transaction log mirror for an existing database any time the database is not running, by using the transaction log utility. This option is available from either Sybase Central or the *dblog* command-line utility.

- From Sybase Central, the transaction log mirror option is part of the Change Log File utility.

  ☞ For more information, see the Sybase Central online Help.

- The following command line starts a transaction log mirror for a database named *company.db*, which is already using one transaction log.

  ```
  dblog -m e:\mirr_dir\company.mlg
  c:\db_dir\company.db
  ```

- The following command line stops the *company.db* database from using a transaction log mirror, but continues maintaining a transaction log:

  ```
  dblog -r c:\db_dir\company.db
  ```

- The following command line stops the *company.db* database from using a transaction log mirror or a transaction log:

  ```
  dblog -n c:\db_dir\company.db
  ```

With the transaction log utility you can also alter the name or directory of the transaction log and mirror.

For a full description of *dblog* command-line options, see "Transaction log utility options" on page 106 of the book *Adaptive Server Anywhere Reference Manual*.

Starting a transaction log mirror for a write file

You can choose to maintain a transaction log mirror for a write file when you create the write file using the Write File utility, or at a later time using the Transaction Log utility.

The option to create a transaction log mirror when creating a write file is available from the CREATE WRITEFILE statement, from Sybase Central, or from the *dbwrite* command-line utility.

- From Sybase Central, the transaction log mirror option is part of the Create Write File utility.

  For more information, see the Sybase Central online Help.

- The following command line (which should be entered on one line) creates a write file for a database named *company.db*, which is already using a transaction log. The write file has default extension .WRT, the write file transaction log has the default extension .WLG, and the write file transaction log mirror has the default extension .WML.

  ```
  dbwrite -c -t d:\log_dir\company.wlg -m
  c:\mirr_dir\company.wml c:\db_dir\company.db
  c:\db_dir\company.wrt
  ```

For a full description of *dbwrite* command-line options, see "Write file utility options" on page 122 of the book *Adaptive Server Anywhere Reference Manual*.

You can change the transaction log and log mirror settings of a write file using the transaction log utility, in exactly the same way as described above for a standard database file.

Erasing transaction log mirrors

You can erase transaction log mirrors using the Erase utility in Sybase Central or the *dberase* command-line utility.

The Erase utility is available in Sybase Central as the Erase Database utility or from the command line as the *dberase* utility.

❖ **To delete a mirror log file only:**

- Enter the following command:

  ```
  dberase e:\mirr_dir\company.wml
  ```

❖ **To delete a transaction log file but not its mirror:**

♦ Enter the following command:

```
dberase e:\log_dir\company.log
```

Validating the transaction log on database startup

When a database that is using a mirror starts up, the server carries out a series of checks and automatic recovery operations to confirm that the transaction log and its mirror are not corrupted, and to correct some problems if corruption is detected.

On startup, the server checks that the transaction log and its mirror are identical by carrying out a full comparison of the two files; if they are identical, the database starts as usual. The comparison of log and mirror adds to database startup time.

If the database stopped because of a system failure, it is possible that some operations were written into the transaction log but not into the mirror. If the server finds that the transaction log and the mirror are identical up to the end of the shorter of the two files, the remainder of the longer file is copied into the shorter file. This produces an identical log and mirror. After this automatic recovery step, the server starts as usual.

If the check finds that the log and the mirror are different in the body of the shorter of the two, one of the two files is corrupt. In this case, the database does not start, and an error message is generated saying that the transaction log or its mirror is invalid.

Recovering from a corrupt transaction log or mirror

When a server detects a difference between the transaction log and its mirror in the body of the file, the server does not start. In this case, you must take the following steps before starting the server:

1 Identify which of the two files is corrupt.

2 Copy the correct file over the corrupt file so that you have two identical files again.

3 Restart the server.

When a server detects a difference between the transaction log and its mirror, it has no means of knowing which is intact and which is corrupt.

❖ **To identify which file is corrupt using the database utilities:**

1. Make a copy of the backup of your database file taken at the time the transaction log was started.
2. Run the log translation utility on the transaction log and on its mirror, to see which one generates an error message. (The log translation utility is accessible from Sybase Central or as the *dbtran* command-line utility.)
3. The following command-line translates a transaction log named *asademo.log*, placing the translated output into *asademo.sql*:

   ```
   dbtran asademo.log
   ```

   The translation utility properly translates the intact file, and will report an error while translating the corrupt file.
4. If the *dbtran* test does not identify the incorrect log, you may want to compare the two translated logs (SQL files) to see which one contains an error, or use a disk utility to inspect the two files and detect which is corrupt.
5. Once you have identified the corrupt file, you can copy the intact log file over the corrupt file, and restart the production server.

Backing up your database

Client-side only
This section describes how to back up your database using the Backup utility. You can also use the BACKUP statement, which provides server-side backup direct to tape. The BACKUP statement is a new feature in version 6.0.2.

For more information, see "BACKUP statement" on page 359 of the book *Adaptive Server Anywhere Reference Manual*.

There are two kinds of backups:

- A **full backup** makes a copy of the database file and (optionally) a copy of the transaction log.
- An **incremental backup** makes a copy of the transaction log.

Both full and incremental backups can be carried out online (while the database is running) or offline.

Online backups

Backups can be made without stopping the server. Using the backup utility on a running database is equivalent to copying the database files when the database is not running. In other words, it provides a snapshot of a consistent database, even though the database is being modified by other users.

For a full description of the online backup facility, see "The Backup utility" on page 67 of the book *Adaptive Server Anywhere Reference Manual*.

Offline backups

The server should not be running when you do offline backups by copying database files. Moreover, it should be taken down cleanly.

If you are running a multi-user database, you can use the database server `-t` command line option to shut down at a specified time. This way, you can have your offline backup procedure start late at night automatically.

Performing a full backup

Check the validity of the database

Before doing a full backup, it is a good idea to verify that the database file is not corrupt. File system errors, or errors in any software that you are running on your machine, could corrupt a small portion of the database file without you ever knowing.

With the database you want to check running on a server, execute the Validation utility. For example, you could run the *dbvalid* command-line utility:

```
dbvalid -c "uid=dba;pwd=sql"
```

You can also run the validation utility from Sybase Central or Interactive SQL.

The validation utility scans every record in every table and looks up each record in each index on the table. If the database file is corrupt, you need to recover from your previous backup.

For more information on running the Validation utility, see "The Validation utility" on page 119 of the book *Adaptive Server Anywhere Reference Manual*.

Back up the database files

A full backup is completed offline by copying the database file(s) and optionally the transaction log to the backup media. A full backup should be completed according to a regular schedule that you follow carefully. Once per week works well for many situations.

To do a backup while the server is running, you use the Backup utility. You require DBA authority in order to run the backup utility on a database. The backup utility can be run from Sybase Central, Interactive SQL, or using the *dbbackup* command-line utility.

For more information, see "The Backup utility" on page 67 of the book *Adaptive Server Anywhere Reference Manual*.

For example, you could carry out a full backup of the sample database, held in *path\asademo.db*, to a directory *e:\backup*, using user ID **DBA**, and password **SQL**.

❖ **To complete a full backup:**

- Enter the following command line:

  ```
  dbbackup -c "uid=dba;pwd=sql;dbf=path\asademo.db"
  e:\backup
  ```

  where *path* is the name of your Adaptive Server Anywhere installation directory.

  As neither `-d n`or `-t` is specified, both the database files and transaction log are backed up.

Transaction log options

Whenever the database file is backed up, the transaction log can be archived and/or deleted (with the Erase utility). If the backup can be restored, you will never need the transaction log. Archiving transaction logs provides you with a history of all changes to your database and also provides protection if you are unable to restore the most recent full backup. The backup utility has command line options to delete and restart the transaction log (*dbbackup* `-x`) or back up and restart the transaction log (*dbbackup* `-r`) while the server is running.

Keep several full backups

You should keep several previous full backups. If you back up on top of the previous backup, and you get a media failure in the middle of the backup, you are left with no backup at all. You should also keep some of your full backups offsite to protect against fire, flood, earthquake, theft, or vandalism.

If your transaction log tends to grow to an unmanageable size between full backups, you should consider getting a larger storage device or doing full backups more frequently.

Performing an incremental backup

An incremental backup is a copy of the transaction log. The transaction log has all changes since the most recent full backup.

You can carry out an offline incremental backup by making a copy of the transaction log file. Alternatively, you can carry out an online incremental backup by running the backup utility and backing up just the transaction log. You can do this from the command line using the *dbbackup* utility with the `-t` switch, or you can use the backup utility from Sybase Central or Interactive SQL. You require DBA authority in order to run the backup utility on a database file.

For example, you could carry out an incremental backup of the sample database, held in *path\asademo.db*, to a directory *e:\backup*, with user ID **DBA**, and password **SQL**.

❖ **To complete an incremental backup:**

- Enter the following command line:

  ```
  dbbackup -c "uid=dba;pwd=sql; dbf=path\asademo.db" -
  t e:\backup
  ```

  where *path* is the name of your Adaptive Server Anywhere installation directory.

Daily backups of the transaction log

If you are running a server that holds critical information, you should back up the transaction log daily. This is particularly important if you have the transaction log on the same device as the database file. If you get a media failure, you could lose both files. By doing daily backups of the transaction log, you will never lose more than one day of changes.

Daily backups of the transaction log are also recommended if the transaction log tends to grow to an unmanageable size between full backups and you do not want to get a larger storage device or do more frequent full backups. In this case, you can choose to archive and delete the transaction log.

There is a drawback to deleting the transaction log after a daily backup. If you have media failure on the database file, there will be several transaction logs since the last full backup. Each of the transaction logs needs to be applied in sequence to bring the database up to date.

For a description of how to do this, see "Media failure on the database file" on page 572.

Recovery from system failure

After a power failure or other system failure you should run the system disk verification program:

- For NetWare: Load the Novell *VREPAIR NLM* to repair any volume that will not mount due to errors.
- For UNIX: Use *chkfsys*
- For Windows 3.x, Windows 95, and Windows NT, run the console command:

```
chkdsk /f
```

This fixes simple errors in the file system structure that might have been caused by the system failure. This should be done before running any other software.

After a system error occurs, the server recovers automatically when you restart the database. The results of each transaction committed before the system error are intact. All changes by transactions that were not committed before the system failure are canceled. It is possible to recover uncommitted changes manually (see "Recovering uncommitted changes" on page 574).

Steps to recover from a system failure

Adaptive Server Anywhere automatically takes three steps to recover from a system failure:

1. Restore all pages to the most recent checkpoint, using the checkpoint log.
2. Apply any changes made between the checkpoint and the system failure. These changes are in the transaction log.
3. Roll back all uncommitted transactions, using the rollback logs. There is a separate rollback log for every connection.

Frequent checkpoints make recovery quicker, but also create work for the server writing out dirty pages.

There are two database options that allow you to control the frequency of checkpoints. CHECKPOINT_TIME controls the maximum desired time between checkpoints and RECOVERY_TIME controls the maximum desired time for recovery in the event of system failure. The RECOVERY_TIME specifies an estimate for steps 1 and 2 only.

☞ For more information on these options, see "General database options" on page 132 of the book *Adaptive Server Anywhere Reference Manual*.

Step 3 may take a long time if there are long uncommitted transactions that have already done a great deal of work since the last checkpoint.

The transaction log is optional. When you are running with no transaction log, a checkpoint is done whenever any transaction is committed. In the event of system failure, the server uses steps 1 and 3 from above to recover a database file. Step 2 is not necessary because there will be no committed transactions since the last checkpoint. This is, however, usually a slower way to run because of the frequent checkpoints.

Using a live backup for machine redundancy

You carry out a live backup of the transaction log by using the *dbbackup* command line utility with the `-l` command-line option. Live backups provide a redundant copy of the transaction log that is available for restart of your system on a secondary machine in case the machine running the database server becomes unusable.

A live backup runs continuously, terminating only if the server shuts down. If you suffer a system failure, the backed up transaction log can be used for a rapid restart of the system.

❖ **To use a live backup:**

1. Periodically, carry out a backup of the database file to a secondary machine.
2. Run a live backup of the transaction log to the secondary machine.
3. If the primary machine becomes unusable, you can restart your database using the secondary machine. The database file and the transaction log hold the information needed to restart.

   To restart, you must first start the database server with the apply transaction log (`-a`) switch, to apply the transaction log and bring the database up to date:

   ```
   dbeng6 asademo.db -a asademo.log
   ```

   The database server shuts down automatically once the transaction log is applied. You can then start the database in the normal way. The server will come up normally, and any new activity will be appended to the current transaction log.

Live backups and transaction log mirrors

There are several differences between using a live backup and using a transaction log mirror:

- **In general, a live backup is made to a different machine** Running a transaction log mirror on a separate machine can lead to performance problems, and will stop the database server if the connection between the machines goes down.

  By running the backup utility on a separate machine, the database server does not do the writing of the backed up log file. Therefore, performance impact is less.

- **A live backup provides protection against a machine becoming unusable** Even if a transaction log mirror is kept on a separate device, it does not provide immediate recovery if the whole machine becomes unusable.

- **A live backup may lag behind the database server** A mirrored transaction log contains all the information required for complete recovery of committed transactions. Depending on the load that the server is processing, the live backup may lag behind and may not contain all the committed transactions.

Live backups and regular backups

The live backup of the transaction log is always the same length or shorter than the active transaction log. When a live backup is running, and another backup restarts the transaction log (dbbackup -r or dbbackup -x), the live backup automatically truncates the live backup log and restarts the live backup at the beginning of the new transaction log.

Recovery from media failure

If you have backups, you can always recover all transactions that were committed before the media failure. Recovery from media failure requires you to keep the transaction log on a separate device from the database file. The information in the two files is redundant. Regular backups of the database file and the transaction log reduce the time required to recover from media failures.

The first step in recovering from a media failure is to clean up, reformat, or replace the device that failed.

The steps to take in recovery depend on whether the media failure is on the device holding your database file or on the device holding your transaction log.

Media failure on the database file

If your transaction log is still usable, but you have lost your database file, the recovery process depends on whether you delete the transaction log on incremental backup.

If you have a single transaction log

If you have not deleted or restarted the transaction log since the last full backup, the transaction log contains everything since the last backup. Recovery involves four steps:

1. Make a backup of the transaction log immediately. The database file is gone, and the only record of the changes is in the transaction log.
2. Restore the most recent full backup (the database file).
3. Use the server with the apply transaction log (`-a`) switch, to apply the transaction log and bring the database up to date:

   ```
   dbeng6 asademo.db -a asademo.log
   ```

4. Start the database in the normal way. The server will come up normally, and any new activity will be appended to the current transaction log.

If you have multiple transaction logs

If you have archived and deleted the transaction log since the last full backup, each transaction log since the full backup needs to be applied in sequence to bring the database up to date.

❖ **To restart from a backup, with multiple transaction logs:**

1. Make a backup of all transaction logs immediately. The database file is gone, and the only record of the changes is in the transaction logs.
2. Restore the most recent full backup (the database file).

3 Starting with the first transaction log after the full backup, apply each archived transaction log by starting the server with the Apply Transaction Log (`-a`) switch. For example, if the last full backup was on Sunday and the database file is lost during the day on Thursday.

```
dbeng6 asademo.db -a sun.log
dbeng6 asademo.db -a mon.log
dbeng6 asademo.db -a tue.log
dbeng6 asademo.db -a wed.log
dbeng6 asademo.db -a sample.log
```

4 Do not apply the transaction logs in the wrong order or skip a transaction log in the sequence.

5 Start the database in the normal way. The server will come up normally, and any new activity will be appended to the current transaction log.

Media failure on the transaction log

If your database file is still usable but you have lost your transaction log, the recovery process is as follows:

1 Make a backup of the database file immediately. The transaction log is gone, and the only record of the changes is in the database file.

2 Restart the database with the `-f` switch.

```
dbeng6 asademo.db -f
```

Without the switch, the server will complain about the lack of a transaction log. With the switch, the server will restore the database to the most recent checkpoint and then roll back any transactions that were not committed at the time of the checkpoint. A new transaction log will be created.

Consequences of media failure on the transaction log

Media failure on the transaction log can have more serious consequences than media failure on the database file. If hen you lose the transaction log, all changes since the last checkpoint are lost. This will be a problem if you have a system failure and a media failure at the same time (such as if a power failure causes a head crash that damages the disk). Frequent checkpoints minimize the potential for lost data, but also create work for the server writing out dirty pages.

For running high-volume or extremely critical applications, you can protect against media failure on the transaction log by mirroring the transaction log or by using a special-purpose device, such as a storage device that mirrors the transaction log automatically. If you are using the server for NetWare, NetWare allows you to automatically mirror a NetWare volume.

For information on using a transaction log mirror, see "Using a transaction log mirror" on page 560.

Recovering uncommitted changes

The transaction log keeps a record of all changes made to the database. Even uncommitted changes are stored in the transaction log. The *dbtran* utility has a command line option (`-a`) to translate transactions that were not committed. With this option, you can recover changes that were not committed by editing the SQL command file and picking out changes that you want to recover.

The transaction log may or may not contain changes right up to the point where a failure occurred. It certainly contains any changes that were made before the most recent COMMIT by *any* transaction.