CHAPTER 21

Oracle8i Backup

In This Chapter

Running in ARCHIVELOG mode

Backing up a database

Protecting control files and redo log files

Testing the integrity of backup files

One of your most important responsibilities as a DBA is to ensure that regular backups are performed and that you can use them to recover the database in the event of a hardware or software failure. You are the person responsible for ensuring that the loss of a disk drive or the accidental deletion of a file doesn't cause permanent loss of data. To fulfill this responsibility, you have to understand how to back up your database. You need to understand such concepts as archive logging so that you can perform up-to-the-minute recovery if data is lost. You also have to understand how to protect your control files and the importance of protecting your database redo log. In this chapter, you'll learn about ARCHIVELOG mode and why it is important. You'll also learn the difference between online and offline backups.

Running in ARCHIVELOG Mode

Most Oracle production databases are run in ARCHIVELOG mode. You'll hear that term a lot, so you need to know just what it means. *ARCHIVELOG mode* is what enables you to recover from a failure without losing any transactions that were committed prior to the time when the failure occurred. In the real world, this is almost always a requirement.

ARCHIVELOG mode is a mode wherein a copy is made of each redo log file before the log writer process reuses it. Recall from Chapter 3, "Oracle8i Architecture," that the log writer cycles through the redo log files in a circular fashion, writing first to one, then to another, and so on, for as long as the database is running. Figure 21-1 illustrates this.

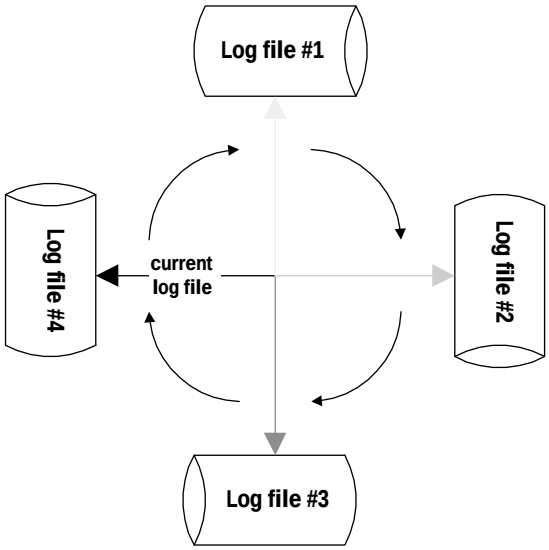


Figure 21-1: Redo log files are written to in a circular fashion.

Each time that the log writer makes another cycle through the redo log files, the previous contents of those files are overwritten. When used this way, you are said to be running in NOARCHIVELOG mode, and the redo log files provide protection only from instance or system crashes. Oracle makes sure that the redo log files always contain enough information to redo any changes that haven't yet been written to the datafiles.

Redo log files can also provide you with the means to recover a lost database, but only if you have saved all the redo log files generated since the most recent full backup. To ensure that all the redo log files are saved, you need to switch your database into ARCHIVELOG mode. When running in ARCHIVELOG mode, Oracle won't overwrite a redo log file until a permanent copy of that file has been created. The copy is called an archive log file, or sometimes an archived log file. Figure 21-2 shows the effect of running a database in archivelog mode.

When you run a database in ARCHIVELOG mode, you can choose to manually make copies of each redo log file as it is filled, or you can start an optional archiver process to automate that copying. Most DBAs end up starting the archiver process and going the automated route.

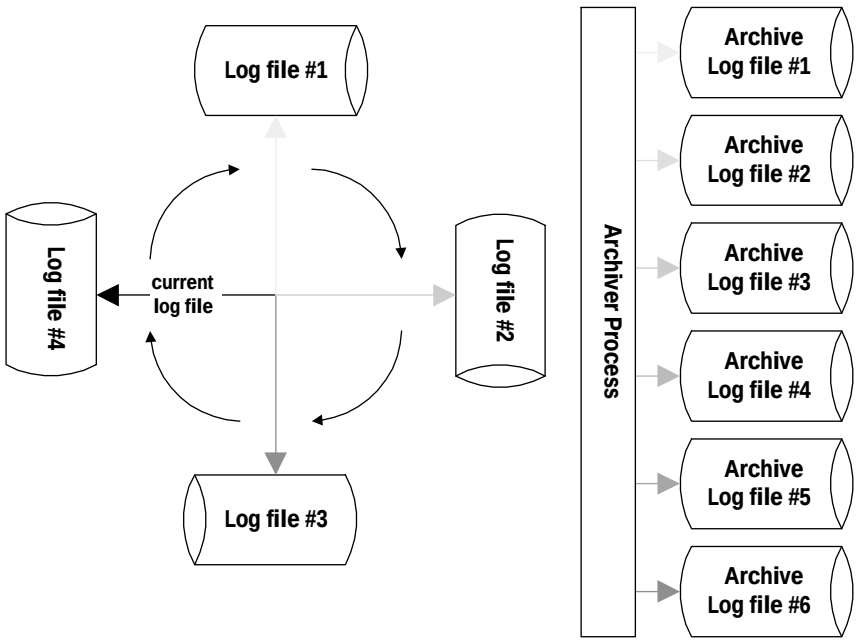


Figure 21-2: Running a database in archivelog mode

Issuing the ARCHIVE LOG LIST command

You can check the archiving status of your database by connecting with SQL*Plus or Server Manager and issuing the `ARCHIVE LOG LIST` command. Consider this example:

```
SQL> ARCHIVE LOG LIST
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            d:\oradata\jonathan
Oldest online log sequence     303
Next log sequence to archive   306
Current log sequence           306
```

In this case, the database is running in ARCHIVELOG mode, and automatic archival has been enabled. That means an archiver process (usually named ARC0) is running in the background that automatically makes a copy of each redo log file as it is filled. The archiver process in this example is currently caught up because the values for "Current log sequence" and "Next log sequence to archive" are the same. The log sequence number is incremented each time Oracle fills one log file and switches to another.

Enabling ARCHIVELOG mode

To enable ARCHIVELOG mode on a database, you can do the following:

1. Set the archive log destination. This is the directory to which archived log files are copied.
2. Define the format of the archive log file names.
3. Check other archive log-related settings.
4. Place your database into archivelog mode.
5. Start the archiver process.

To place a database into ARCHIVELOG mode, you have to close it. Since you're likely to be changing several initialization file parameters, it's usually easiest to *bounce* the instance — in other words, to shut it down and start it up again.

Setting the Archive Log Destination

The archiver process needs to know where to place the archive log files that it creates. Oracle8i allows you to specify up to five archive log destinations. As each redo log file is filled, the archiver process will copy it to each destination that you specify. You use the following initialization parameters to specify the destinations:

log_archive_dest_1

log_archive_dest_2

log_archive_dest_3

log_archive_dest_4

log_archive_dest_5

For most cases, one archive log destination is enough. The destination should be the full path of a directory somewhere on disk. The following example shows an entry from a database parameter file for `log_archive_dest_1`:

```
log_archive_dest_1 = "location=d:\oradata\jonathan"
```

The `location` keyword is part of the specification, and it indicates that a directory name follows. You can use the other destination parameters, `log_archive_dest_2`, `log_archive_dest_3`, and so forth, as you choose. Setting them to a null string ("") is the same as omitting them from the parameter file altogether.

Older releases of Oracle supported only one archive log destination, which was specified by the `log_archive_dest` parameter (no digit in the parameter name). The "`location=`" is not used when setting that parameter. Oracle8i supports either method, but you have to do everything one way or the other. Use either `log_archive_dest` or `log_archive_dest_1`, `log_archive_dest_2`, and so forth. You can't mix the two methods.

When you change your parameter file, you need to stop and restart the database for that change to take effect. If your database is running, you can use the `ALTER SYSTEM` command to temporarily change the archive log destination. Issue the command like this:

```
ALTER SYSTEM SET archive_log_dest_1 = 'LOCATION=path'
```

Replace *path* with the full path to the directory in which you want to place the archive log files. When you make a change using `ALTER SYSTEM` like this, remember that the next time the database starts, it will use whatever archive log destination is set in the parameter file. For changes to be permanent, they must be made to the parameter file.

Defining the Format of Archive Log File Names

Archive log files need to have names, and Oracle lets you control the format of those names. You specify the name format through the `log_archive_format` parameter. That parameter must be set to a string of characters that represents a valid file name on your system. You can use two special character sequences to make archive log file names unique. These character sequences are the following:

- `%s` — Gets replaced by the log sequence number, which increments with each file
- `%t` — Gets replaced by the redo log thread number

The "`%s`" string is a must have. It must be part of your archive log file name to make each file name unique. The "`%t`" string gets replaced by a thread number and is applicable to Oracle Parallel Server (OPS). Under OPS, each instance gets its own thread of redo. If two or more instances share an archive log destination, include the thread number as part of the redo log file name.

It's wise to always use `%t` and `%s` in the archive log file name.

The following example shows the parameter setting that you can use if you want the archive log files to start with `arch` and to contain the sequence number and thread numbers separated by underscores:

```
log_archive_format = "arch_%s.%t"
```

File names generated using this parameter setting take this form:

```
arch_1.1
arch_2.1
arch_3.1
...
```

You can use the `ALTER SYSTEM` command to change the `log_archive_format` parameter while the database is running, but such a change won't be permanent. Stopping and restarting the database will cause the archiver to revert back to the `log_archive_format` specified in the parameter file.

Placing Your Database in ARCHIVELOG Mode

Once you've defined an archive log destination and you've defined the archive log file name format, you are ready to place your database into archive log mode. You do that using the `ALTER DATABASE ARCHIVELOG` command. Your database must be closed to make this change. Because you're likely to be making initialization parameter changes at the same time that you are placing your database into ARCHIVELOG mode, it's usually best to shut down and restart everything. Listing 21-1 shows an example of a database being placed into archivelog mode:

Listing 21-1: Placing a database into ARCHIVELOG mode

```
SVRMGR> CONNECT / AS SYSDBA
Connected.
SVRMGR> SHUTDOWN
Database closed.
Database dismounted.
ORACLE instance shut down.
SVRMGR> STARTUP MOUNT
ORACLE instance started.
Total System Global Area                38322124 bytes
Fixed Size                                 65484 bytes
Variable Size                           21405696 bytes
Database Buffers                        16777216 bytes
Redo Buffers                               73728 bytes
Database mounted.
SVRMGR> ALTER DATABASE ARCHIVELOG;
Statement processed.
SVRMGR> ALTER DATABASE OPEN;
Statement processed.
SVRMGR>
```

Here, the database was first shut down and then restarted. This causes any new settings for archive log-related parameters to take effect. You start the database using the MOUNT keyword, which prevents it from being opened. You use the ALTER DATABASE ARCHIVELOG command to place the database into ARCHIVELOG mode. Finally, you use the ALTER DATABASE OPEN command to open the database for general use. If you like, you can issue the ARCHIVE LOG LIST command to verify the new ARCHIVELOG status.

Starting the Archiver Process

If you don't want to manually archive each log file, and most people don't, you must start Oracle's archiver process. There are two ways to do this. One way is to issue the ARCHIVE LOG START command each time that you start the database. The other way is to set the log_archive_start initialization parameter.

You can issue the ARCHIVE LOG START command from SQL*Plus or Server Manager whenever you're logged on as INTERNAL, SYSDBA, or SYSOPER. The syntax is simple, and there is no output to speak of. You simply issue the command as shown in the following example:

```
SVRMGR> ARCHIVE LOG START;
Statement processed.
```

A corresponding ARCHIVE LOG STOP command also exists. If you would like the archiver process to start automatically whenever you start the database, you can place the following setting into your initialization parameter file:

```
log_archive_start = true
```

With this in place, every time you start the database, Oracle will automatically start the archiver process.

Managing archive log files

When you run a database in ARCHIVELOG mode, you must periodically purge old archive log files. Otherwise, the directory that is your archive log destination will fill up, the archiver won't be able to copy any more log files, and your database will grind to a halt. You must make two decisions when it comes to purging old archive log files:

- How long do you want to keep the files?
- How long do you want to keep the files online where they can be conveniently accessed?

At a minimum, you need to keep all the archive log files since your most recent full backup. If that's too many to keep online at once, you can periodically copy them to tape and delete them from disk, thus making room for more files. It's easier if you can manage to keep them all online. That way, restoring a datafile doesn't force you into having to retrieve a bunch of old archive log files from tape. Most DBAs end up writing shell scripts to periodically copy old archive log files to tape and delete them from disk.

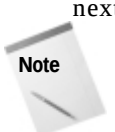
Archiving a log file manually

If you're running the database in ARCHIVELOG mode but you don't have the archiver process running, then you will need to manually archive your log files. You can't do that just by copying them. You have to archive them in such a way that Oracle knows that they have been copied. That way, Oracle knows when it's safe to reuse a log file and when it isn't.

You can use the `ARCHIVE LOG` command to manually archive log files. You would normally use two forms of the command. The `ARCHIVE LOG ALL` command tells Oracle to copy all log files that have been filled but that haven't been copied yet. The `ARCHIVE LOG NEXT` command tells Oracle to archive just the one log file that is next in the sequence. The following example shows the `ARCHIVE LOG ALL` command being used to archive all filled redo log files that haven't been archived yet:

```
SVRMGR> archive log all
1 log archived.
SVRMGR>
```

Why would you want to manually archive log files, as opposed to letting the archiver do it for you automatically? If you're archiving to disk, it's probably better to let the archiver do it automatically. However, if you are archiving to tape or some other removable media device, then you may want to exert control over when the redo log files are copied. Having the archiver on automatic in such a case would require that you leave your tape (or other media) mounted all the time. It might not be feasible to do that. If you're manually archiving, then you only need to mount the tape while you issue the `ARCHIVE LOG ALL` command. Then you can dismount the tape until next time.



If you are manually archiving, be sure to have enough redo log files and size them large enough so that you can go a reasonable length of time before having to archive. Consider sizing your redo log files to hold a day's worth of data. That way, you have to archive only once per day. Remember, if your redo log files all fill up, your database will stop.

Backing Up a Database

Backing up an Oracle database involves copying the files that make up the database to a backup storage medium. There are two types of backups to be aware of:

- Online backups, often referred to as *hot backups*
- Offline backups, often referred to as *cold backups*

The difference between an online and an offline backup is that you perform an online backup when the database is open; you perform an offline backup when the database is completely shut down — that is, closed and dismounted.

An Oracle database is composed of several types of files. When you back up a database, you should be sure to back up the following files:

- All the datafiles
- The control file (if an online backup is being performed, you can't just copy this file)
- The database initialization parameter file
- The archive log files

The only other file type you'll encounter is the online redo log file. Some DBAs back these up as well, but it's a dangerous practice. Oracle recommends against it. The reason you shouldn't back up redo log files is that there's really no reason to ever have to restore them. In fact, generally, if you do restore your online redo logs, you will lose data. If you are recovering a database, and you want to recover the database to the way it was at the moment that it was lost, you need the most current redo log files to do that. If you restore an older set of redo log files on top of your current files, you've just lost whatever data was in those files. You should protect redo log files by multiplexing, not by backing them up. The sidebar discusses the danger of restoring redo log files.

You should back up control files, but you still need to be careful about restoring them. If you are performing an offline backup, you can just copy your control files to tape. If you are performing an online backup, then Oracle has the control files open and you need to issue an `ALTER DATABASE` command to specify to make the copy for you.

Datafiles, initialization parameter files, and archive log files are always backed up by copying the files. If you are performing an online backup, you do have to specify when you are copying the datafiles.

The Danger in Restoring Redo Log Files

In the heat of battle, it's easy to slip up and restore the wrong files. When restoring from an online backup, if you want to roll forward and bring the database up to date, you generally don't want to restore either the control files or the redo log files. I was once going through a practice run of a large database restore where I did just that. There's nothing quite like that sinking feeling when you realize you've just shot yourself in the foot and trashed your database. Fortunately, it was only a test.

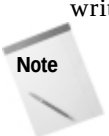
Backing up a database offline

Offline backups are the simplest to implement. You shut down the database, copy the files to tape, and then restart the database. Simple as they are to implement, offline backups do carry with them one big disadvantage: They require the database to be closed. In many shops today, 24x7 availability is required, so shutting down the database for a backup isn't an option.

If you can afford the downtime to do an offline backup each night, then that's probably the best way to go. Certainly, it's the simplest. Otherwise, you need to go with the approach of performing online backups. The process for performing an offline backup is simple. Just follow these steps:

1. Shut down the database.
2. Copy the datafiles, control files, initialization files, and archive log files to tape.
3. Restart the database.

In general, you will want to write scripts to automate this process. On small systems, such as an NT server, the offline backup can be part of the nightly operating system backup. To make that work, you need to write pre- and post-backup scripts to shut down the database before the backup and to restart it afterwards. Commercial backup software, such as Seagate's Backup Exec or Sterling Software's SAMS Alexandria, is capable of automatically executing scripts before and after a backup. Oracle's Recovery Manager Software (often referred to as RMAN) can be used to write backup scripts as well.



If you are scheduling unattended offline backups, it is critical to periodically check to be sure that your scripts are in fact shutting down the database first. Many times, DBAs assume this is happening, when in fact it isn't.

Backing up a database online

Online backups, as the name suggests, are those performed while the database is open. That is also their primary advantage. Doing an online backup is a much more complex process than doing an offline backup. Ultimately, you end up copying the datafiles while the database is open. However, you must tell Oracle that you are doing so; otherwise, the backup won't be valid.

To perform online backups, your database must be running in ARCHIVELOG mode because restoring from an online backup always involves recovering transactions from the database log files. You have to be archiving redo log files to be sure of having them when you need them.

To perform an online backup, the procedure gets more complicated. Generally, you need to do the following:

1. Back up the control files.
2. Back up the datafiles for some or all of the tablespaces.
3. Archive the current online redo log files.
4. Back up the archive log files.

The first three steps require you to issue commands to the database. Step 2 can be especially complicated because you need to intersperse SQL statements with your file copies. You can write your own scripts to automate online backups, or you can use one of the commercial backup tools on the market. Oracle supplies Recovery Manager (RMAN), which is a utility that you can use to automate the backup process. Several third-party vendors also provide backup solutions that work with Oracle. One such vendor is Sterling Software (`http://www.sterling.com`), which markets an automated backup and tape management software package called SAMS Alexandria.

Backing Up Control Files

When you perform an online backup, by definition, the database is open. If you just copied the open control file using operating system commands, you wouldn't get a consistent version of the file.

When the database is open and you want to back up the control file, you need to issue an `ALTER DATABASE` command like this:

```
ALTER DATABASE BACKUP CONTROLFILE  TO 'filename';
```

Replace *filename* with the full directory path and file name to which you want Oracle to copy the control file.

In addition to making a binary copy of the control file, it can also be helpful to generate a file containing the SQL statements necessary to re-create one from scratch. Oracle will do that for you too. Just issue the following command:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

When you issue this command, Oracle will write the commands necessary to re-create the control file to a trace file. Finding the trace file can be a bit tricky. Oracle writes it to your database's user_dump_dest directory (usually $ORACLE_BASE/admin/sid/udump) and names it ORAxxxx.trc, where xxxxx is a five-digit number. To find the correct file in the user_dump_dest directory, you have to look at the file creation dates. Find the file or files created at about the time you issued the command — hopefully, there will be only one — and open each one to see if they contain a `CREATE CONTROL` file statement.

Having a trace file with commands to rebuild your control file comes in handy if you ever need to move your datafiles or if you are restoring your database and placing the files in different directories than they were in originally.

Backing Up Datafiles

When you perform an online backup of a database, you need to copy the datafiles one tablespace at a time. Before copying the files for a tablespace, you need to specify that you are doing so by issuing the following command:

```
ALTER TABLESPACE tablespace_name BEGIN BACKUP;
```

Replace *tablespace_name* with the name of the tablespace whose files you are going to copy. When you are done copying files for the tablespace, issue the following command:

```
ALTER TABLESPACE tablespace_name END BACKUP;
```

This specifies that you are done backing up the tablespace. Repeat this process for each tablespace that you back up. The reason for these `BEGIN BACKUP` and `END BACKUP` commands is that Oracle needs to maintain the datafile headers in a consistent state while they are being copied. When you issue the `BEGIN BACKUP`, Oracle stops updating the checkpoint in the file headers of the affected datafiles. During the time that the tablespace is in backup mode, Oracle records changes to the data in that tablespace by writing entire data blocks to the redo log files. In normal operation, Oracle just writes enough information to describe each change.

As an example, if you had a database with three tablespaces — SYSTEM, USERS, and TOOLS — your online backup sequence would look like this:

1. Issue an `ALTER TABLESPACE system BEGIN BACKUP` command.
2. Copy the files for the SYSTEM tablespace.

3. Issue an `ALTER TABLESPACE system END BACKUP` command.
4. Issue an `ALTER TABLESPACE users BEGIN BACKUP` command.
5. Copy the files for the USERS tablespace.
6. Issue an `ALTER TABLESPACE users END BACKUP` command.
7. Issue an `ALTER TABLESPACE tools BEGIN BACKUP` command.
8. Copy the files for the TOOLS tablespace.
9. Issue an `ALTER TABLESPACE tools END BACKUP` command.

Any script to do this will have to constantly be switching in and out of either Server Manager or SQL*Plus. You need to issue the `ALTER TABLESPACE` commands from one of those tools, but you need to issue the file copies by using operating system commands.

You also have to deal with the fact that the datafiles in a database change occasionally. If you had a shell script to do your online backups, you would need to modify the shell script each time a datafile was added or removed. Another option is to write one shell script that dynamically generates one or more other scripts based on information in the data dictionary. Because this can get complex, many sites will purchase a third-party tool to automate the backup process.

If you are not using an automated backup and recovery solution already, you might want to consider Oracle's Recovery Manager. It is discussed in the sidebar.

Recovery Manager

If you're not already using an automated backup and recovery solution, take a look at Oracle's Recovery Manager. Recovery Manager was introduced with Oracle8, and it has the following capabilities:

- ✦ It can automate the process of performing both offline and online backups.
- ✦ It compresses backups by only copying database blocks that are being used.
- ✦ It can optionally maintain a recovery catalog, contained in an Oracle database other the one being backed up, which records each backup and restore operation that you do.
- ✦ It has the ability to interface to third-party tape management software.
- ✦ It can perform incremental backups, backing up only data that has changed since the most recent backup prior to the one being taken.

Recovery Manager is a command-line utility, and some of the commands get rather complex, but Oracle Enterprise Manager implements a set of wizards that builds and executes Recovery Manager command files for you.

Archiving the Current Online Redo Log Files

After you finish backing up all the datafiles, you need to archive the current online redo log files because they are needed for recovery. Archiving them allows them to be backed up with all the other archive log files. To have Oracle archive the current redo log files, issue this command:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

This command causes Oracle to switch to a new log file. Oracle then archives all redo log files that have not yet been archived. Another approach to this same task is to issue the following two commands:

```
ALTER SYSTEM SWITCH LOGFILE;
ALTER SYSTEM ARCHIVE LOG ALL
```

The first command forces the log switch, and the second causes Oracle to archive all the redo log files that are full but that haven't been archived yet.

Backing Up Archive Log Files

Once you've archived your current online redo log files, the last step is to back up all of your archive log files. You'll need these to restore the database. You don't need to issue any special commands to Oracle to back up these files. Just copy them to tape using whatever commands are appropriate for your operating system.

Exporting databases

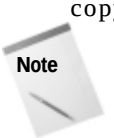
Database exports can be considered a form of backup. However, you should always consider exports a supplement to the other types of backup. Exports are good for restoring an object that you drop accidentally or for restoring the definition of such an object, but that's about it. The process for restoring an entire database from an export is cumbersome. Unlike a file-based backup, you can't use an export to recover a database up to the point of failure. When you restore from an export, you get only what is in the export file. You can't roll forward from that. Exports are not substitutes for database file backups.

Protecting Control Files and Redo Log Files

Control files and redo log files are necessary for recovery. You always need to have the latest versions of the files available, so you can't protect them by copying them to tape. Instead, you need to protect yourself from ever losing these files, and you do that by multiplexing them. To *multiplex* a file means to keep two or more copies of the file, and to always write the same information to both.

You can multiplex files using hardware, software, or a combination of both. Hardware solutions involve disk mirroring. You configure two disks so that the same information is always written to both. If one disk goes bad, the other disk takes

over, and you don't lose any data. To multiplex using software, you specify that you have multiple files, and the Oracle software takes care of writing everything to each copy of the file.



If you depend totally on hardware mirroring, be aware that a corrupt write may end up corrupting the mirror as well as the primary copy of the file.

If you're particularly worried about losing data (and most DBAs probably are a bit paranoid about this), you can combine both methods. If you currently are not multiplexing your control and redo log files, you should begin doing so.

Multiplexing control files

If you want to increase the number of control file copies for your database, you need to follow these steps:

1. Shut down the database.
2. Copy one of the control files to a new location, on a separate disk drive.
3. Modify the `control_files` parameter in your database parameter file so that the new control file is included in the list of control files for your database.
4. Restart your database.

You actually do have to go through a shutdown-and-restart process to add a new control file. You can't use the `ALTER DATABASE` statement to make a copy of the control file while performing this procedure, as you could when you were just making a backup, because once the `ALTER DATABASE` command finishes, the control file copy rapidly gets out of sync with the original.

The `control_files` parameter in the initialization file lists all the control files for the database. Oracle looks at this list when you start a database. The parameter setting consists of a comma-delimited list of file names, and it looks like this:

```
control_files = ("E:\Oracle\oradata\jonathan\control01.ctl",
                 "F:\Oracle\oradata\jonathan\control02.ctl")
```

When you start a database, Oracle reads this parameter, opens the files that are listed, and writes the same information to each of the files as long as the database is mounted. If you're not sure how many control files you have in your current database, you can find out by querying `the V$CONTROLFILE` dynamic performance view. Consider this example:

```
SQL> SELECT * FROM v$controlfile;

STATUS  NAME
------- ----------------------------------------
        E:\ORACLE\ORADATA\JONATHAN\CONTROL01.CTL
        F:\ORACLE\ORADATA\JONATHAN\CONTROL02.CTL
```

The STATUS field should normally be null. If it contains the string INVALID, then you have a problem with that control file. Always maintain at least two control files (three is better) for a database, and they should all be on separate disks. That way, the loss of any one disk doesn't compromise your database.

Multiplexing redo log files

You can add more redo log files to your database while it is running. You may recall from Chapter 3, "Oracle8i Architecture," that redo log files are organized into groups, each group having multiple members. Oracle writes the same information to all members of a group.

You can query the V$LOGFILE view to see how many redo log groups you have and to see how many members (files) are in each group. Consider this example:

```
SQL> SELECT * FROM v$logfile
  2  ORDER BY group#;

   GROUP# STATUS  MEMBER
--------- ------- ---------------------------------------
        1         E:\ORACLE\ORADATA\JONATHAN\REDO04.LOG
        2         E:\ORACLE\ORADATA\JONATHAN\REDO03.LOG
        3         E:\ORACLE\ORADATA\JONATHAN\REDO02.LOG
        4         E:\ORACLE\ORADATA\JONATHAN\REDO01.LOG
```

If you are running Oracle Parallel Server, you can query the GV$LOGFILE view. It contains the same information as V$LOGFILE, but it also identifies the instance using each log file.

In this example, there are four groups, with one log file in each group. Unless hardware mirroring is being used, this is not an ideal situation. Loss of the E drive would result in the loss of data. You can add files to a redo log group by issuing the following ALTER DATABASE command:

```
ALTER DATABASE
   ADD LOGFILE MEMBER
      'filename'
   TO GROUP group_number;
```

Replace *group_number* with the number of the group to which you want to add a member. Replace *filename* with the fully qualified path and file name that you want to use for the new log file. Oracle will create the new log file and add it to the group. The four commands shown in Listing 21-2 add a log file to each of the four groups shown previously.

Listing 21-2: **Adding log files to groups**

```
ALTER DATABASE
   ADD LOGFILE MEMBER
      'F:\ORACLE\ORADATA\JONATHAN\REDO04.LOG'
   TO GROUP 1;

ALTER DATABASE
   ADD LOGFILE MEMBER
      'F:\ORACLE\ORADATA\JONATHAN\REDO03.LOG'
   TO GROUP 2;

ALTER DATABASE
   ADD LOGFILE MEMBER
      'F:\ORACLE\ORADATA\JONATHAN\REDO02.LOG'
   TO GROUP 3;

ALTER DATABASE
   ADD LOGFILE MEMBER
      'F:\ORACLE\ORADATA\JONATHAN\REDO01.LOG'
   TO GROUP 4;
```

There will now be two members per group, and the output from querying the `V$LOGFILE` view will look like this:

```
SQL> SELECT * FROM v$logfile
  2  ORDER BY group#;

   GROUP# STATUS  MEMBER
--------- ------- ----------------------------------------
        1         E:\ORACLE\ORADATA\JONATHAN\REDO04.LOG
        1         F:\ORACLE\ORADATA\JONATHAN\REDO04.LOG
        2         E:\ORACLE\ORADATA\JONATHAN\REDO03.LOG
        2         F:\ORACLE\ORADATA\JONATHAN\REDO03.LOG
        3         E:\ORACLE\ORADATA\JONATHAN\REDO02.LOG
        3         F:\ORACLE\ORADATA\JONATHAN\REDO02.LOG
        4         F:\ORACLE\ORADATA\JONATHAN\REDO02.LOG
```

With two log files per group, and each file on a different drive, there is no longer a single point of failure.

Testing the Integrity of Backup Files

You can test the integrity of your backup files in a couple of different ways. One way is to use Oracle's DBVERIFY utility to see whether or not the physical structure of the data blocks within a backup datafile looks good. Another way to validate the integrity of your backup files is to attempt to restore them.

Using the DBVERIFY utility

The DBVERIFY utility is a command-line utility. The command to start it is dbv. Under Windows NT, with releases prior to 8i, the command will be dbverf80, dbverf73, and so forth. You can invoke the DBVERIFY utility by using the following syntax:

```
dbv parameter=value [parameter=value...]
```

The *parameter* element must be one of the items in the list that follows. The *value* element in the syntax should be replaced with a value appropriate for the parameter with which it's associated. The value for the FILE parameter, for example, should be a file name.

- FILE — Specifies the name of the database file to verify.
- START — Specifies the starting block number to verify. If START is not specified, DB_VERIFY assumes the first block in the file.
- END — Specifies the ending block number to verify. If END is not specified, DB_VERIFY assumes the last block in the file.
- BLOCKSIZE — Specifies the block size. A size of 2,048 is assumed if you don't specify otherwise.
- LOGFILE — Specifies the file to which DBVERIFY output should be written. (The default sends output to the terminal display.)
- FEEDBACK — Specifies whether DBVERIFY should display its progress on the screen while DB_VERIFY runs. The feedback is in the form of a dot that is displayed for every *n* blocks, where *n* is the value that you supply. For example, FEEDBACK=1000 results in a dot (.) being displayed for every 100 database blocks.
- HELP — Allows you to get help on these parameters. Use HELP=Y.
- PARFILE — Allows you to read DBVERIFY parameters from a file. This functions just like PARFILE does when used with the Import and Export utilities.

The example in Listing 21-3 shows DBVERIFY being run against a file named item_data.dbf. The feedback parameter has been set to 100, so a dot will display for each 100 blocks (or pages) that have been examined.

The DBVERIFY utility uses the term *pages* to mean blocks. A page and a database block are the same thing.

Listing 21-3: **Running DBVERIFY**

```
$dbv file=item_data.dbf feedback=100 blocksize=8096

DBVERIFY: Release 8.1.5.0.0 - Production on Thu Oct 7 15:57:52

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

DBVERIFY - Verification starting : FILE = item_data.dbf
.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

.....................................................................

.........................


DBVERIFY - Verification complete

Total Pages Examined         : 51200
Total Pages Processed (Data) : 0
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 1
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 5
Total Pages Empty            : 51194
Total Pages Marked Corrupt   : 0
Total Pages Influx           : 0
```

You can run the DBVERIFY utility only against datafiles. It can't check redo log files or archive log files.

Testing the restore process

One of the best favors you can do for yourself is to test your ability to restore your database from a backup. Although you can't do a trial restore on every backup that you take, you can certainly do a trial run once in awhile. If you can actually restore the database, then you know that all your database files are being backed up properly, that your `ALTER TABLESPACE BEGIN BACKUP` and `ALTER TABLESPACE END BACKUP` commands are being executed when they should, and so forth. Not only will a few trial restores bolster your confidence in your backup process, but they will also give you valuable practice for the real thing.

Summary

In this chapter, you learned:

- One of the most important tasks, if not *the* most important task, you'll need to do is to back up your database. It's critical that you plan for the eventual hardware and software problems that will cause you to lose datafiles, and that you ensure that you are always in a position to recover from those problems. No one else will worry about this for you.
- To enable up-to-the-minute recovery of a database, you must be running that database in archivelog mode, and you must save all the archived log files since the most recent full backup. That way, if you lose a file, you can restore it from the backup and then bring it up to date by applying any changes to it that have been recorded in the redo log files.
- Offline backups are those performed when the database is completely shut down. The process for doing them is simple. Shut down the database. Copy the files to tape. Restart the database.
- Online backups are those performed while the database is running. You must use the `ALTER DATABASE BACKUP CONTROLFILE` command to copy the control file. You must issue an `ALTER TABLESPACE ... BEGIN BACKUP` command before copying the files for a tablespace, and you must issue an `ALTER TABLESPACE ... END BACKUP` command afterwards. You must archive all online redo log files, which you can do by issuing an `ALTER SYSTEM ARCHIVE LOG CURRENT` command, and then you must back up all your archive log files.
- Redo log files and control files are critical to recovery, and you need to be sure that you always have the latest versions of those files available. Therefore, you should protect redo log files and control files by multiplexing them. Redo log files should not normally be backed up with the database files.
- If you're in doubt about the integrity of a backup datafile, you can run `DBVERIFY` against the file. The `DBVERIFY` utility does a check of the internal structure of the data blocks within the file, and it reports any problems that it finds.

✦ ✦ ✦