15 Backup and Recovery Concepts

CERTIFICATION OBJECTIVES

Perhaps the most important aspect of a database administrator's job is to ensure that the database does not lose data. The mechanisms of redo and undo ensure that it is impossible to corrupt the database no matter what the DBA does, or does not, do (always assuming that there is no physical damage). After working through the section of this chapter headed Instance Recovery, you will be able to prove this. However, it is possible for an Oracle database to lose data if the DBA does not take appropriate precautions.

From release 9*i* onward, an Oracle database can be configured so that no matter what happens, the database will never lose a single row of committed data. It is also possible to configure an environment for one hundred percent availability. This ideal configuration requires use of Data Guard and RAC (or possibly Streams). A single-instance, non-distributed environment cannot achieve this—but it can get very close.

This chapter will go through the concepts behind Oracle's backup and recovery mechanisms: the enabling structure within which you will configure whatever level of data security and availability is demanded by your organization. The next two chapters will cover the practicalities of backup, restore, and recovery. But always be aware that this is a superficial treatment; the second OCP examination deals with backup and recovery in much greater detail. And even after passing that, you will not necessarily be fully competent. This is an area where you cannot do enough studying, research, and (most importantly) practice.

CERTIFICATION OBJECTIVE 15.01

Identify the Types of Failure That Can Occur in an Oracle Database

Failures can be divided into a few broad categories. For each type of failure, there will be an appropriate course of action to resolve it. Each type of failure may well be documented in a service level agreement; certainly the steps to be followed should be documented in a procedures manual.

Statement Failure

An individual SQL statement can fail for a number of reasons, not all of which are within the DBA's domain—but even so, he/she must be prepared to fix them. The first level of fixing will be automatic. Whenever a statement fails, the server process

executing the statement will detect the problem and roll back the statement. Remember that a statement might attempt to update many rows, and fail part way through execution; all the rows that were updated before the failure will have their changes reversed through use of undo. This will happen automatically. If the statement is part of a multistatement transaction, all the statements that have already succeeded will remain intact, but uncommitted. Ideally, the programmers will have included exceptions clauses in their code that will identify and manage any problems, but there will always be some errors that get through the error handling routines.

A common cause of statement failure is invalid data, usually a format or constraint violation. A well-written user process will avoid format problems, such as attempting to insert character data into a numeric field, but they can often occur when doing batch jobs with data coming from a third-party system. Oracle itself will try to solve formatting problems by doing automatic typecasting to convert data types on the fly, but this is not very efficient and shouldn't be replied upon. Constraint violations will be detected, but Oracle can do nothing to solve them. Clearly, problems caused by invalid data are not the DBA's fault, but you must be prepared to deal with them by working with the users to validate and correct the data, and with the programmers to try to automate these processes.

A second class of non-DBA-related statement failure is logic errors in the application. Programmers may well develop code that in some circumstances is impossible for the database to execute. A perfect example is the deadlock described in Chapter 10: the code will run perfectly, until through bad luck two sessions happen to try do the same thing at the same time to the same rows. A deadlock is not a database error; it is an error caused by programmers writing code that permits an impossible situation to arise.

Space management problems are frequent, but they should never occur. A good DBA will monitor space usage proactively and take action before problems arise. Space-related causes of statement failure include inability to extend a segment because the tablespace is full, running out of undo space, insufficient temporary space when running queries that use disk sorts or working with temporary tables, a user hitting his/her quota limit, or an object hitting its maximum extents limit. Database Control includes the undo advisor, the segment advisor, the Automatic Database Diagnostic Monitor, and the alert mechanism, all described in previous chapters, which will help to pick up space-related problems before they happen. The effect of space problems that slip through can perhaps be alleviated by setting datafiles to autoextend, or by enabling resumable space allocation, but ideally space problems should never arise in the first place.

on the Job

Issue the command "alter session enable resumable" and from then on the session will not show errors on space problems but instead hang until the problem is fixed. You can enable resumable for the whole instance with the RESUMABLE_TIMEOUT parameter.

exam Watch

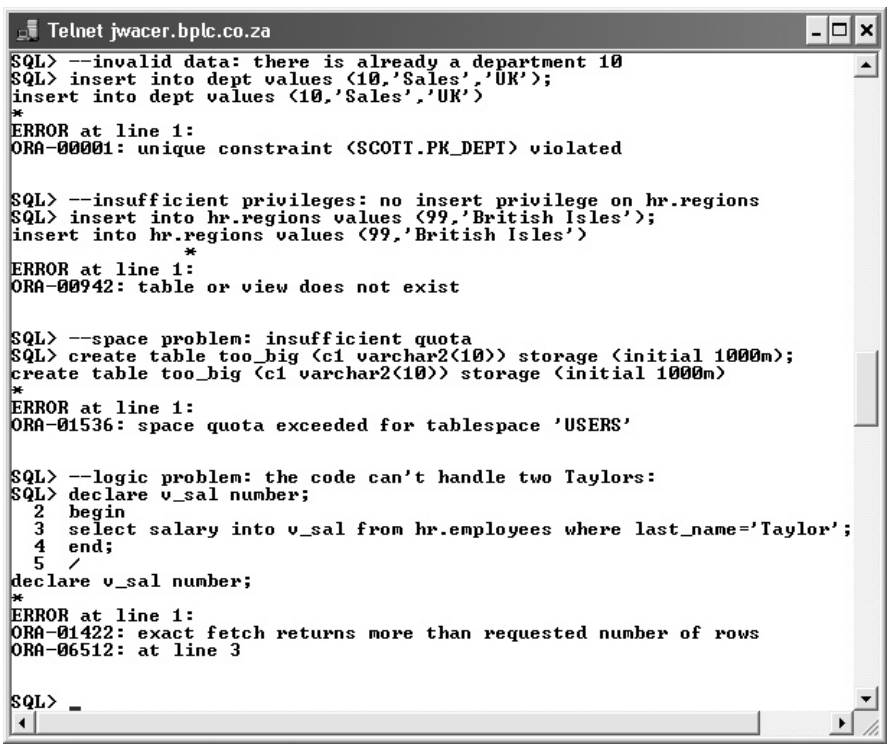
If a statement fails, it will be rolled back. Any other DML statements will remain intact and uncommitted.

Statements may fail because of insufficient privileges. Remember from Chapter 8 how privileges let a user do certain things, such as select from a table or execute a piece of code. When a statement is parsed, the server process checks whether the user executing the statement has the necessary permissions. This type of error indicates that the security structures in place are inappropriate, and the DBA (in conjunction with the organization's security manager) should grant appropriate system and object privileges.

Figure 15-1 shows some examples of statement failure: a data error, a permissions error, a space error, and a logic error.

FIGURE 15-1

Examples of statement failures



User Process Failure

A user process may fail for any number of reasons, including the user exiting abnormally instead of logging out, the terminal rebooting, or the program causing an address violation. Whatever the cause of the problem, the outcome is the same. The PMON background process periodically polls all the server processes, to ascertain the state of the session. If a server process reports that it has lost contact with its user process, PMON will tidy up. If the session were in the middle of a transaction, PMON will roll back the transaction and release any locks. Then it will terminate the server process and release the PGA back to the operating system.

exam Watch

If a session terminates abnormally, an active transaction will be rolled back automatically.

This type of problem is beyond the DBA's control, but he/she should watch for any trends that might indicate a lack of user training, badly written software, or perhaps network or hardware problems.

Network Failure

In conjunction with the network administrators, it should be possible to configure Oracle Net such that there is no single point of failure. The three points to consider are listeners, network interface cards, and routes.

A database listener is unlikely to crash, but there are limits to the amount of work that one listener can do. A listener can service only one connect request at a time, and it does take an appreciable amount of time to launch a server process and connect it to a user process. If your database experiences high volumes of concurrent connection requests, users may receive errors when they try to connect. You can avoid this by configuring multiple listeners, each on a different address/port combination.

At the operating system and hardware levels, network interfaces can fail. Ideally, your server machine will have at least two network interface cards, for redundancy as well as performance. Create at least one listener for each card.

Routing problems or localized network failures can mean that even though the database is running perfectly, no one can connect to it. If your server has two or more network interface cards, they should ideally be connected to physically separate subnets. Then on the client side configure connect time fault tolerance by listing multiple addresses in the ADDRESS_LIST section of the TNS_NAME.ORA entry. This will permits the user processes to try a series of routes until they find one that is working.

on the job

The network fault tolerance for a single-instance database is only at connect time; a failure later on will disrupt currently connected sessions, and they will have to reconnect. In a RAC environment, it is possible for a session to fail over to a different instance, and the user may not even notice.

User Errors

Historically, *user errors* were undoubtedly the worst errors to manage. Recent releases of the database improve the situation dramatically. The problem is that user errors are not errors as far as the database is concerned. Imagine a conversation on these lines:

User: "I forgot to put a WHERE clause on my UPDATE statement, so I've just updated a million rows instead of one."
DBA: "Did you say COMMIT?"
User: "Of course."
DBA: "Um . . ."

As far as Oracle is concerned, this is a transaction like any other. The *D* for "Durable" of the ACID test states that once a transaction is committed, it must be immediately broadcast to all other users, and be absolutely non-reversible. But at least with DML errors such as the one dramatized here, the user does get the chance to roll back his/her statement if he realizes that it was wrong before committing. But DDL statements don't give you that option. For example, if a programmer drops a table believing he/she is logged onto the test database but is actually logged onto the production database, there is a COMMIT built into the DROP TABLE command. That table is gone—you can't roll back DDL.

on the job

Never forget that there is a COMMIT built into DDL statements that will include any preceding DML statements.

The ideal solution to user errors is to prevent their occurring in the first place. This is partly a matter of user training, but more importantly of software design: no user process should ever let a user issue an UPDATE statement without a WHERE clause. But even the best-designed software cannot prevent users from issuing SQL that is inappropriate to the business. Everyone makes mistakes. Oracle provides a number of ways whereby you as DBA may be able to correct user errors, but this is often extremely difficult—particularly if the error isn't reported for some time. The possible techniques (details of which are beyond the scope of this book) include

flashback query, flashback drop, the Log Miner, incomplete recovery, and flashback database.

Flashback query is running a query against a version of the database as at some time in the past. The read-consistent version of the database is constructed, for your session only, through use of undo data.

Figure 15-2 shows one of many uses of flashback query. The user has "accidentally" deleted every row in the EMP table, and committed the delete. Then he/she retrieves the rows with a subquery against the table as it was five minutes previously.

Flashback drop reverses the effect of a DROP TABLE command. In previous releases of the database, a DROP command did what it says: it dropped all references to the table from the data dictionary. There was no way to reverse this. Even flashback query would fail, because the flashback query mechanism does need the data dictionary object definition. But from release 10g the implementation of the DROP command has changed: it no longer drops anything; it just renames the object so that you will never see it again, unless you specifically ask to. Figure 15-3 illustrates the use of flashback drop to recover a table.

The Log Miner is an advanced tool that extracts information from the online and archived redo logs. Redo includes all changes made to data blocks. By extracting the changes made to blocks of table data, it is possible to reconstruct the changes that were made—thus, redo can be used to bring a restored backup forward in time.

FIGURE 15-2

Correcting user error with flashback query

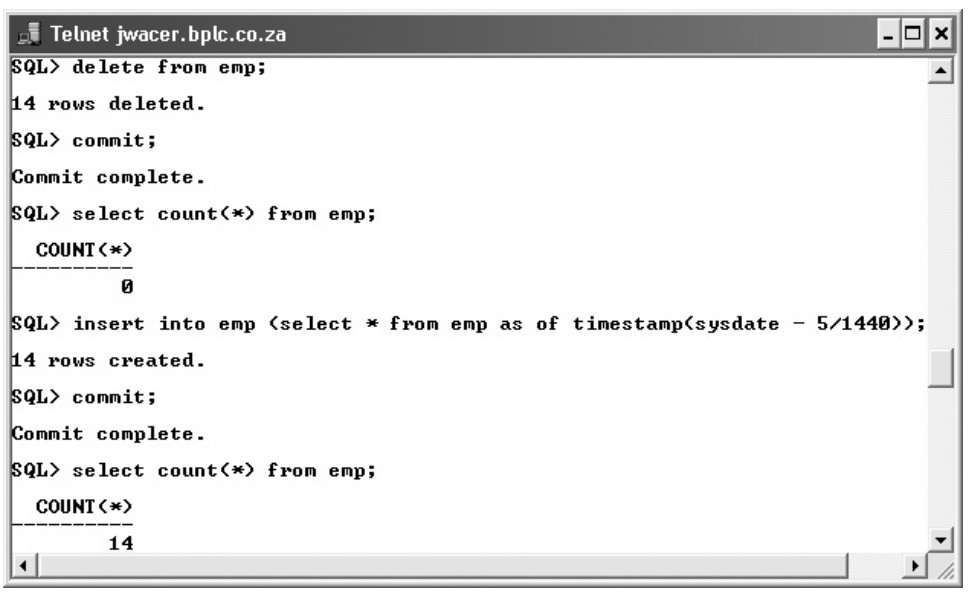
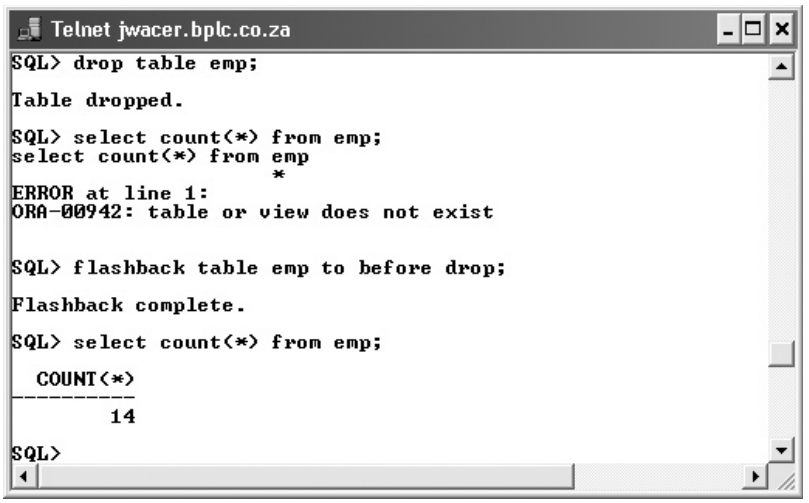


FIGURE 15-3

Correcting user error with flashback drop



But the redo stream also has all the changes made to undo blocks, and it is therefore possible to construct the changes that would be needed to reverse transactions, even though they have been committed. Conceptually, the Log Miner is similar to flashback query: the information to reverse a change is constructed from undo data, but whereas flashback query uses undo data that is currently in the undo segments, Log Miner extracts the undo data from the redo logs. This means that Log Miner can go back in time indefinitely, if you have copies of the relevant logs. By contrast, flashback query can only go back as far as your undo tablespace will allow.

Incomplete recovery and flashback database are much more drastic techniques for reversing user errors. With either tool, the whole database is taken back in time to before the error occurred. The other techniques that have been described let you reverse one bad transaction, while everything else remains intact. But if you ever do an incomplete recovery, or a flashback of the whole database, you will lose all the work done from time you go back to—not just the bad transaction.

Media Failure

Media failure means damage to disks, and therefore the files stored on them. This is not your problem (it is something for the system administrators to sort out), but you must be prepared to deal with it. The point to hang on to is that damage to any number of any files is no reason to lose data. With release 9*i* and later, you can survive the loss of any and all of the files that make up a database without losing any committed data—if

you have configured the database appropriately. Prior to 9*i*, complete loss of the machine hosting the database could result in loss of data; the Data Guard facility, not covered in the OCP curriculum, can even protect against that.

Included in the category of "media failure" is a particular type of user error: system or DBAs accidentally deleting files. This is not as uncommon as one might think (or hope).

On Unix systems, the `rm` command has been responsible for any number of appalling mistakes. You might want to consider, for example, aliasing the `rm` command to `rm -i` to gain a little piece of mind.

When a disk is damaged, one or more of the files on it will be damaged, unless the disk subsystem itself has protection through RAID. Remember that a database consists of three file types: the control file, the online redo logs, and the datafiles. The control file and the online logs should always be protected through multiplexing. If you have multiple copies of the control file on different disks, then if any one of them is damaged, you will have a surviving copy. Similarly, multiple copies of each online redo log mean that you can survive the loss of any one. Datafiles can't be multiplexed (other than through RAID, at the hardware level); therefore, if one is lost the only option is to *restore* it from a backup. To restore a file is to extract it from wherever it was backed up, and put it back where it is meant to be. Then the file must be *recovered*. The restored backup will be out of date; recovery means applying changes extracted from the redo logs (both online and archived) to bring it forward to the state it was in at the time the damage occurred.

Recovery requires the use of archived redo logs. These are the copies of online redo logs, made after each log switch. After restoring a datafile from backup, the changes to be applied to it to bring it up-to-date are extracted, in chronological order, from the archive logs generated since the backup was taken. Clearly, you must look after your archive logs because if any are lost, the recovery process will fail. Archive logs are initially created on disk, and because of the risks of using disk storage they, just like the controlfile and the online log files, should be multiplexed: two or more copies on different devices.

So to protect against media failure, you must have multiplexed copies of the controlfile, the online redo log files, and the archive redo log files. You will also take backups of the controlfile, the data files, and the archive log files. You do not back up the redo logs—they are, in effect, backed up when they are copied to the archive logs. Datafiles cannot be protected by multiplexing; they need to be protected by hardware redundancy—either conventional RAID systems, or Oracle's own Automatic Storage Management (ASM).

Instance Failure

An *instance failure* is a disorderly shutdown of the instance, popularly referred to as a crash. This could be caused by a power cut, by switching off or rebooting the server machine, or by any number of critical hardware problems. In some circumstances, one of the Oracle background processes may fail—this will also trigger an immediate instance failure. Functionally, the effect of an instance failure, for whatever reason, is the same as issuing the SHUTDOWN ABORT command. You may hear people talking about "crashing the database" when they mean issuing a SHUTDOWN ABORT command.

After an instance failure, the database may well be missing committed transactions and storing uncommitted transactions. This is the definition of a corrupted database. This situation arises because the server processes work in memory: they update blocks of data and undo segments in the database buffer cache, not on disk. DBWn then, eventually, writes the changed blocks down to the datafiles. The algorithm the DBWn uses to select which dirty buffers to write is oriented toward performance and results in the blocks that are least active getting written first—after all, there would be little point in writing a block that is being changed every second. But this means that at any given moment there may well be committed transactions that are not yet in the datafiles and uncommitted transactions that have been written: there is no correlation between a COMMIT and a write to the datafiles. But of course, all the changes that have been applied to both data and undo blocks are already in the redo logs.

Remember the description of commit processing detailed in Chapter 10: when you say COMMIT, all that happens is that LGWR flushes the log buffer to the current online redo log files. DBWn does absolutely nothing on COMMIT. So for performance reasons, DBWn writes as little as possible as rarely as possible—this means that the database is always out of date. But LGWR writes with a very aggressive algorithm indeed. It writes as nearly as possible in real time, and when you (or anyone else) say COMMIT, it really does write in real time. This is the key to instance recovery. Oracle accepts the fact that the database will be corrupted after an instance failure, but there will always be enough information in the redo log stream on disk to correct the damage.

EXERCISE 15-1

Correct Statement Failures

In this exercise, you will install demonstrate the automatic rollback of statements after failures.

1. Connect to the database as user SYSTEM with SQL*Plus, and create a table to use for this exercise:

```
create table failures as select * from all_users where 1=2;
```

2. Insert some rows (don't COMMIT) and check the number:

```
insert into failures select * from all_users;
select count(*) from all_users;
```

3. Using either Database Control or another SQL*Plus session, connect to the database as SYS and terminate the instance with ABORT.
4. Your SYSTEM session will have been ended. Reconnect, and count the rows in the FAILURES table. The INSERT from Step 2 will have been rolled back.
5. Repeat Step 2.
6. Terminate the SQL*Plus user process: use the Unix or Linux kill command, or on Windows use the task manager.
7. Reconnect as SYSTEM: again, your INSERT will have been rolled back.
8. Tidy up:

```
drop table failures;
```

CERTIFICATION OBJECTIVE 15.02

Describe Ways to Tune Instance Recovery

The rules to which a relational database must conform, as formalized in the ACID test, require that it may never lose a committed transaction and never show an uncommitted transaction. Oracle conforms to the rules perfectly. If the database is corrupted, Oracle will detect the fact and perform instance recovery to remove the corruptions. It will reinstate any committed transactions that had not been saved to the datafiles at the time of the crash, and roll back any uncommitted transactions that had been written to the datafiles. This instance recovery is completely automatic—you can't stop it, even if you wanted to. If the instance recovery fails, which will only happen if there is media failure as well as instance failure, you cannot open the database until you have used media recovery techniques to restore and recover the damaged files. The final step of media recovery is automatic instance recovery.

The Mechanics of Instance Recovery

Because instance recovery is completely automatic, it can be dealt with fairly quickly, unlike media recovery, which will take a whole chapter just for the simplest technique. In principle, instance recovery is nothing more than using the contents of the online log files to rebuild the database buffer cache to the state it was in before the crash. This will replay all changes extracted from the redo logs that refer to blocks that had not been written to disk at the time of the crash. Once this has been done, the database can be opened. At that point, the database is still corrupted—but there is no reason not to allow users to connect, because the instance (which is what users see) has been repaired. This phase of recovery, known as the roll forward, reinstates all changes—changes to data blocks and changes to undo blocks—for both committed and uncommitted transactions. Each redo record has the bare minimum of information needed to reconstruct a change: the block address, and the new values. During roll forward, each redo record is read, the appropriate block loaded from the datafiles into the database buffer cache, and the change is applied. Then the block is written back to disk.

Once the roll forward is complete, it is as though the crash had never occurred. But at that point, there will be uncommitted transactions in the database—these must be rolled back, and Oracle will do that automatically in the rollback phase of instance recovery. However, that happens after the database has been opened for use. If a user connects and hits some data that needs to be rolled back and hasn't yet been, this is not a problem—the roll forward phase will have populated the undo segment that was protecting the uncommitted transaction, so the server can roll back the change in the normal manner for read consistency.

Instance recovery is automatic, and unavoidable—so how do you invoke it? By issuing a STARTUP command. Remember from Chapter 5, on starting an instance, the description of how SMON opens a database. First, it reads the controlfile when the database transitions to mount mode. Then in the transition to open mode, SMON checks the file headers of all the datafiles and online redo log files. At this point, if there had been an instance failure, it is apparent because the file headers are all out of sync. So SMON goes into the instance recovery routine, and the database is only actually opened after the roll forward phase has completed.

You never have anything to lose by issuing a STARTUP command. After any sort of crash, try a STARTUP and see how far it gets. It might get all the way.

The Impossibility of Database Corruption

It should now be apparent that there is always enough information in the redo log stream to reconstruct all work done up to the point at which the crash occurred, and furthermore that this includes reconstructing the undo information needed to roll back transactions that were in progress at the time of the crash. But for the final proof, consider this scenario.

User JOHN has started a transaction. He has updated one row of a table with some new values, and his server process has copied the old values to an undo segment. But before these updates were done, his server process wrote out the changes to the log buffer. User ROOPESH has also started a transaction. Neither has committed; nothing has been written to disk. If the instance crashed now, there would be no record whatsoever of either transaction, not even in the redo logs. So neither transaction would be recovered—but that is not a problem. Neither was committed, so they should not be recovered: uncommitted work must never be saved.

Then user JOHN commits his transaction. This triggers LGWR to flush the log buffer to the online redo log files, which means that the changes to both the table and the undo segments for both JOHN's transaction and ROOPESH's transaction are now in the redo log files, together with a commit record for JOHN's transaction. Only when the write has completed is the "commit complete" message returned to JOHN's user process. But there is still nothing in the datafiles. If the instance fails at this point, the roll forward phase will reconstruct both the transactions, but when all the redo has been processed, there will be no commit record for ROOPESH's update; that signals SMON to roll back ROOPESH's change but leave JOHN's in place.

But what if DBWR has written some blocks to disk before the crash? It might be that JOHN (or another user) was continually requerying his data, but that ROOPESH had made his uncommitted change and not looked at the data again. DBWn will therefore decide to write ROOPESH's changes to disk in preference to JOHN's; DBWn will always tend to write inactive blocks rather than active blocks. So now, the datafiles are storing ROOPESH's uncommitted transaction but missing JOHN's committed transaction. This is as bad a corruption as you can have. But think it through. If the instance crashes now—a power cut, perhaps, or a SHUTDOWN ABORT—the roll forward will still be able to sort out the mess. There will always be enough information in the redo stream to reconstruct committed changes; that is obvious, because a commit isn't completed until the write is done. But because LGWR flushes *all* changes to *all* blocks to the log files, there will also be enough information to reconstruct the undo segment needed to roll back ROOPESH's uncommitted transaction.

exam Watch

Can a SHUTDOWN ABORT corrupt the database? Absolutely not! It is impossible to corrupt the database.

So to summarize, because LGWR always writes ahead of DBWn, and because it writes in real time on commit, there will always be enough information in the redo stream to reconstruct any committed changes that had not been written to the datafiles, and to roll back any uncommitted changes that had been written to the data files. This instance recovery mechanism of redo and rollback makes it absolutely impossible to corrupt an Oracle database—so long as there has been no physical damage.

Tuning Instance Recovery

A critical part of many service level agreements as the MTTR—the *mean time to recover* after various events. Instance recovery guarantees no corruption, but it may take a considerable time to do its roll forward before the database can be opened. This time is dependent on two factors: how much redo has to be read and how many read/write operations will be needed on the datafiles as the redo is applied. Both these factors can be controlled by checkpoints.

A checkpoint guarantees that as of a particular time, all data changes made up to a particular SCN, or System Change Number, have been written to the datafiles by DBWn. In the event of an instance crash, it is only necessary for SMON to replay the redo generated from the last checkpoint position. All changes, committed or not, made before that position are already in the datafiles; so clearly, there is no need to use redo to reconstruct the transactions committed prior to that. Also, all changes made by uncommitted transactions prior to that point are also in the datafiles—so there is no need to reconstruct undo data prior to the checkpoint position either; it is already available in the undo segment on disk for the necessary rollback.

The more up-to-date the checkpoint position is, the faster the instance recovery. If the checkpoint position is right up-to-date no roll forward will be needed at all—the instance can open immediately and go straight into the rollback phase. But there is a heavy price to pay for this. To advance the checkpoint position, DBWn must write changed blocks to disk. Excessive disk I/O will cripple performance. But on the other hand, if you let DBWn get too far behind, so that after a crash SMON has to process hundreds of megabytes of redo and do millions of read/write operations on the datafiles, the MTTR following an instance failure can stretch into hours.

Tuning instance recovery time used to be largely a matter of experiment and guesswork. It has always been easy to tell how long the recovery actually took—just look at your alert log, and you will see the time when the STARTUP command was issued and the time that the startup completed, with information about how

many blocks of redo were processed—but until release 9*i* of the database it was almost impossible to calculate in advance. Release 9*i* introduced a new parameter, FAST_START_MTTR_TARGET, that makes controlling instance recovery time a trivial exercise. You specify it in seconds, and Oracle will then ensure that DBWn writes out blocks at a rate sufficiently fast that if the instance crashes, the recovery will take no longer than that number of seconds. So the smaller the setting, the harder DBWn will work in an attempt to minimize the gap between the checkpoint position and real time. But note that it is only a target—you can set it to an unrealistically low value, which is impossible to achieve no matter what DBWn does. Database Control also provides an MTTR advisor, which will give you an idea of how long recovery would take if the instance failed. This information can also be obtained from the view V$INSTANCE_RECOVERY.

The MTTR Advisor and Checkpoint Auto-Tuning

The parameter FAST_START_MTTR_TARGET defaults to zero. This has the effect of maximizing performance, with the possible cost of long instance recovery times after an instance failure. The DBWn process will write as little as it can get away with, meaning that the checkpoint position may be a long way out of date and that therefore a large amount of redo would have to be applied to the datafiles in the roll forward phase of instance recovery.

Setting FAST_START_MTTR_TARGET to a non-zero value has two effects. First, it sets a target for recovery, as described in the preceding section. But there is also a secondary effect: enabling *checkpoint auto-tuning*. The checkpoint auto-tuning mechanism inspects statistics on machine utilization, such as the rate of disk I/O and CPU usage, and if it appears that there is spare capacity, it will use this capacity to write out additional dirty buffers from the database buffer cache, thus pushing the checkpoint position forward. The result is that even if the FAST_START_MTTR_TARGET parameter is set to a high value (the highest possible is 3600 seconds—anything above that will be rounded down), actual recovery time may well be much less.

Enabling checkpoint auto-tuning with a high target should result in your instance always having the fastest possible recovery time that is consistent with maximum performance.

Database Control has an interface to the parameter. From the database home page, take the Advisor Central link, and then the MTTR advisor to get a window that displays the current estimated recovery time (this is the advisor) and gives the

option of adjusting the parameter. More complete information can be gained from querying the V$INSTANCE_RECOVERY view, described here:

```
SQL> desc v$instance_recovery;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 RECOVERY_ESTIMATED_IOS                             NUMBER
 ACTUAL_REDO_BLKS                                   NUMBER
 TARGET_REDO_BLKS                                   NUMBER
 LOG_FILE_SIZE_REDO_BLKS                            NUMBER
 LOG_CHKPT_TIMEOUT_REDO_BLKS                        NUMBER
 LOG_CHKPT_INTERVAL_REDO_BLKS                       NUMBER
 FAST_START_IO_TARGET_REDO_BLKS                     NUMBER
 TARGET_MTTR                                        NUMBER
 ESTIMATED_MTTR                                     NUMBER
 CKPT_BLOCK_WRITES                                  NUMBER
 OPTIMAL_LOGFILE_SIZE                               NUMBER
 ESTD_CLUSTER_AVAILABLE_TIME                        NUMBER
 WRITES_MTTR                                        NUMBER
 WRITES_LOGFILE_SIZE                                NUMBER
 WRITES_LOG_CHECKPOINT_SETTINGS                     NUMBER
 WRITES_OTHER_SETTINGS                              NUMBER
 WRITES_AUTOTUNE                                    NUMBER
 WRITES_FULL_THREAD_CKPT                            NUMBER
```

The critical columns are

Column	Meaning
recovery_estimated_ios	The number of read/write operations that would be needed on datafiles for recovery if the instance crashed now
actual_redo_blocks	The number of OS blocks of redo that would need to be applied to datafiles for recovery if the instance crashed now
estimated_mttr	The number of seconds it would take to open the database if it crashed now
target_mttr	The setting of fast_start_mttr_target
writes_mttr	The number of times DBWn had to write, in addition to the writes it would normally have done, to meet the target mttr
writes_autotune	The number of writes DBWn did that were initiated by the auto-tuning mechanism

Tracking the value of the ESTIMATED_MTTR will tell you if you are keeping to your service level agreement for crash recovery time; WRITES_MTTR tells you the price you are paying for demanding a fast recovery time.

exam Watch

Checkpoint auto-tuning is enabled if FAST_START_MTTR_TARGET is set to a non-zero value.

EXERCISE 15-2

Monitor Instance Recovery Times

In this exercise, you will observe the effect of checkpointing on instance recovery times.

1. Connect to your database with SQL*Plus as user SYSTEM.
2. Create a table to use for the exercise:

```
create table cptest as select * from all_users;
```

3. Determine the work involved in recovery, if the instance were to crash now:

```
select recovery_estimated_ios, actual_redo_blks, estimated_mttr
from v$instance_recovery;
```

   Note the figures: they should show a recovery time estimated to be only a few seconds.

4. Start a large transaction by running this statement a dozen or so times:

```
insert into cptest select * from cptest;
```

5. Rerun the query from Step 3. You should see that the figures have increased—though this is a matter of luck, as the database will be attempting to push the checkpoint position forward.
6. COMMIT the transaction, and rerun the query from Step 2. There should be no difference: this demonstrates that a COMMIT does not force DBWn to do anything.
7. Force a checkpoint:

```
alter system checkpoint;
```

8. Rerun the query from Step 2. Now, the figures should have dropped back down to near zero, because the checkpoint forced DBWn to write all dirty buffers to the datafiles.
9. Tidy up by dropping the table:

```
drop table ctptest;
```

SCENARIO & SOLUTION	
It is often said that production databases should run in archivelog mode, and that test and development systems don't need to. Is that right?	People who say that have never lost a development or test system. The idea is that they aren't "real" systems, so if you lose days' work it doesn't matter. Well, if your testing and development team is a dozen consultants at US$2000 a day, it probably does matter.
Databases don't crash very often, so should the DBA really worry about recovery time?	Recovery time is only important if it is important. Many applications can stand downtime, and if the database has crashed, then the part of the downtime that is the instance recovery may not matter. But for some databases, every second is important. Downtime means no transactions, and if no transactions means no revenue, then if tuning the recovery time can save a few minutes, you must do it.

CERTIFICATION OBJECTIVE 15.03

Identify the Importance of Checkpoints, Redo Log Files, and Archived Log Files

Checkpointing is the process of forcing the DBWn to write dirty buffers from the database buffer cache to the datafiles. There are circumstances when this must be done to ensure that the database could be recovered. Also necessary for recovery are the online redo logs, and the archive redo logs.

Checkpointing

As discussed in the preceding section, the checkpoint position (the point in the redo stream from which instance recovery must start following a crash) is advanced automatically by the DBWn. This process is known as *incremental* checkpointing. In addition, there may be *full checkpoints* and *partial checkpoints*.

A full checkpoint occurs when all dirty buffers are written to disk. In normal running, there might be a hundred thousand dirty buffers, but the DBWn would write just a few hundred of them for the incremental checkpoint. For a full checkpoint, it will write the lot. This will entail a great deal of work: very high CPU and disk usage while the checkpoint is in progress, and reduced performance for user sessions. Full checkpoints are bad for business. Because of this, there will never be a full checkpoint except in two circumstances: an orderly shutdown, or at the DBA's request.

When there database is shut down with the NORMAL, IMMEDIATE, or TRANSACTIONAL option, there is a checkpoint: all dirty buffers are flushed to disk by the DBWn before the database is closed and dismounted. This means that when the database is opened again, no recovery will be needed. A clean shutdown is always desirable and is necessary before some operations (such as enabling the archivelog mode or the flashback database capability). A full checkpoint can be signaled at any time with this command:

```
alter system checkpoint;
```

A partial checkpoint is necessary and occurs automatically as part of certain operations. Depending on the operation, the partial checkpoint will affect different buffers. These are

Operation	What Buffers Will Be Flushed from the Cache
Taking a tablespace offline	All blocks that are part of the tablespace
Taking a datafile offline	All blocks that are part of the datafile
Dropping a segment	All blocks that are part of the segment
Truncating a table	All blocks that are part of the table
Putting a tablespace into backup mode	All blocks that are part of the tablespace

exam Watch

Full checkpoints occur only with an orderly shutdown, or by request.

Partial checkpoints occur automatically as needed.

on the job

Manually initiated checkpoints should never be necessary in normal running, though they can be useful when you want to test the effect of tuning. There is no checkpoint following a log switch. This has been the case since release 8i, though to this day many DBAs do not realize this.

Protecting the Online Redo Log Files

Remember that an Oracle database requires at least two online log file groups to function, so that it can switch between them. You may need to add more groups for performance reasons, but two are required. Each group consists of one or more members, which are the physical files. Only one member per group is required for Oracle to function, but at least two members per group are required for safety.

on the job

Always have at least two members in each log file group, for security. This is not just data security—it is job security, too.

The one thing that a DBA is not allowed to do is to lose all copies of the current online log file group. If that happens, you will lose data. The only way to protect against data loss when you lose all members of the current group is to configure a Data Guard environment for zero data loss, which is not a trivial exercise. Why is it so critical that you do not lose all members of the current group? Think about instance recovery. After a crash, SMON will use the contents of the current online log file group for roll forward recovery, to repair any corruptions in the database. If the current online log file group is not available, perhaps because it was not multiplexed and media damage has destroyed the one member, then SMON cannot do this. And if SMON cannot correct corruptions with roll forward, you cannot open the database.

Just as with multiplexed copies of the controlfile, the multiple members of an online log file group should ideally be on separate disks, on separate controllers. But when considering disk strategy, think about performance as well as fault tolerance. In the discussion of commit processing in Chapter 10, it was made clear that when a COMMIT is issued, the session will hang until LGWR has flushed the log buffer to disk. Only then is "commit complete" returned to the user process, and the session allowed to continue. This means that writing to the online redo log files is one of the ultimate bottlenecks in the Oracle environment: you cannot do DML faster than LGWR can flush changes to disk. So on a high-throughput system, make sure that your redo log files are on your fastest disks served by your fastest controllers.

exam Watch

The online redo log can be reconfigured while the database is open with no downtime, whereas operations on the controlfile can only be carried out when the database is in nomount mode or completely shut down.

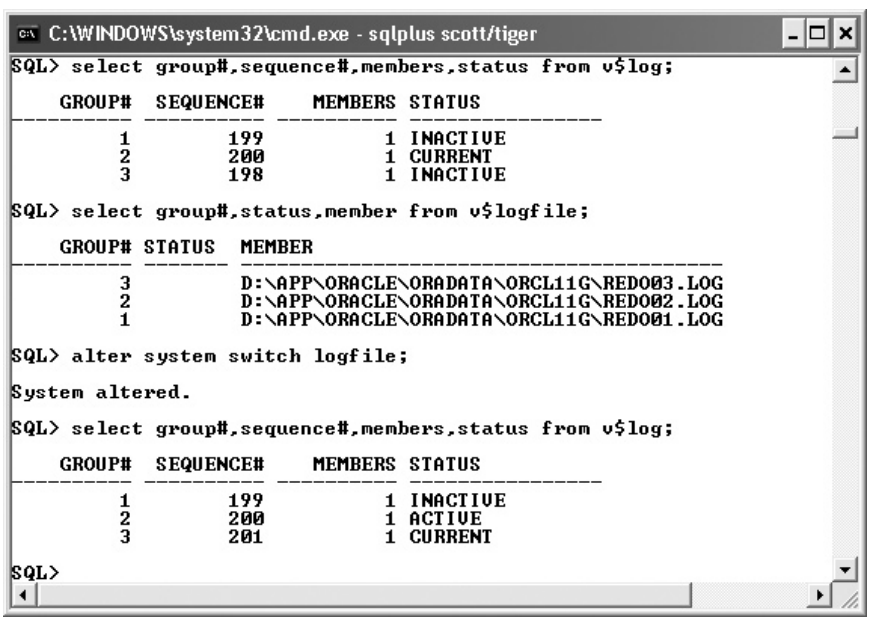
If a member of a redo log file group is damaged or missing, the database will remain open if there is a surviving member. This contrasts with the controlfile, where damage to any copy will crash the database immediately. Similarly, groups can be added or removed and members of groups can be added or moved while the database is open, as long as there are always at least two groups, and each group has at least one valid member.

If you create a database with DBCA, by default you will have three groups, but they will have only one member each. You can add more members (or indeed whole groups) either through Database Control or from the SQL*Plus command line. There are two views that will tell you the state of your redo logs. V$LOG will have one row per group, and V$LOGFILE will have one row per log file member. Figure 15-4 shows an example of online redo log configuration.

The first query shows that this database has three log file groups. The current group—the one LGWR is writing to at the moment—is group 2; the other groups are inactive, meaning first that the LGWR is not writing to them, and second that

FIGURE 15-4

Online redo log file configuration



in the event of an instance failure, SMON would not require them for instance recovery. In other words, the checkpoint position has advanced into group 2. The SEQUENCE# column tells us that there have been 200 log switches since the database was created. This number is incremented with each log switch. The MEMBERS column shows that each group consists of only one member—seriously bad news, which should be corrected as soon as possible.

The second query shows the individual online redo log files. Each file is part of one group (identified by GROUP#, which is the join column back to V$LOG) and has a unique name. The STATUS column should always be null, as shown. If the member has not yet been used, typically because the database has only just been opened and no log switches have occurred, the status will be STALE; this will only be there until the first log switch. If the status is INVALID, you have a problem.

As with the controlfile, Oracle does not enforce any naming convention for log files, but most organizations will have standards for this.

Then there is a command to force a log switch:

```
alter system switch logfile;
```

The log switch would happen automatically, eventually, if there were any DML in progress.

The last query shows that after the log switch, group 3 is now the current group that LGWR is writing to, at log switch sequence number 201. The previously current group, group 2, has status ACTIVE. This means that it would still be needed by SMON for instance recovery if the instance failed now. In a short time, as the checkpoint position advances, it will become INACTIVE. Issuing an

```
alter system checkpoint;
```

command would force the checkpoint position to come up-to-date, and group 2 would then become inactive immediately.

The number of members per group is restricted by settings in the controlfile, determined at database creation time. Turn back to Chapter 4, and the CREATE DATABASE command called by the `CreateDB.sql` script; the MAXLOGFILES directive limits the number of groups that this database can have, and the MAXLOGMEMBERS directive limits the maximum number of members of each group. The DBCA defaults for these (sixteen and three, respectively) may well be suitable for most databases, but if they prove to be inappropriate, it is possible to recreate the controlfile with different values. However, as with all controlfile operations, this will require downtime.

To protect the database against loss of data in the event of damage to an online redo log file group, multiplex it. Following the example in Figure 15-4, to add multiplexed copies to the online log, one would use a command such as this:

```
alter database add logfile member
'D:\APP\ORACLE\ORADATA\ORCL11G\REDO01A.log' to group 1;
```

or it can also be done through Database Control.

Archivelog Mode and the Archiver Process(es)

Oracle guarantees that your database is never corrupted, through use of the online redo log files to repair any corruptions caused by an instance failure. This is automatic and unavoidable. But to guarantee no loss of data following a media failure, it is necessary to have a record of all changes applied to the database since the last backup of the database; this is not enabled by default. The online redo log files are overwritten as log switches occur, so the history of change vectors is by default not kept—but the transition to *archivelog mode* ensures that no online redo log file is overwritten unless it has been copied as an archive log file first. So there will be a series of archive log files that represent a complete history of all changes ever applied to the database. If a datafile is damaged at any time, it will then be possible to restore a backup of the datafile and apply the changes from the archive log redo stream to bring it up-to-date.

exam Watch

An online redo log file can be overwritten only if it is inactive, and (if the database is in archivelog mode) if it has been archived.

By default, a database is created in *noarchivelog mode*; this means that online redo log files are overwritten by log switches with no copy being made first. It is still impossible to corrupt the database, but data would be lost if the datafiles were damaged by media failure. Once the database is transitioned to archivelog mode, it is impossible to lose data—provided that all the archive log files generated since the last backup are available.

Once a database is converted to archivelog mode, a new background process will start, automatically. This is the archiver process, ARCn. By default Oracle will start four of these processes, but you can have up to thirty. In earlier releases of the database it was necessary to start this process either with a SQL*Plus command or by setting the initialization parameter LOG_ARCHIVE_START, but from release 10*g* onward Oracle will automatically start the archiver if the database is in archivelog mode.

on the Job

In archivelog mode, recovery is possible with no loss of data up to and including the last commit. As a general rule, all databases where you cannot afford to lose data should run in archivelog mode. Don't exclude test and development systems from this rule; they are important too.

The archiver will copy the online redo log files to an archive log file after each log switch, thus generating a continuous chain of log files that can be used for recovering a backup. The name and location of these archive log files is controlled by initialization parameters. For safety the archive log files can be multiplexed, just as the online log files can be multiplexed—but eventually, they should be migrated to offline storage, such as a tape library. The Oracle instance takes care of creating the archive logs with the ARCn process, but the migration to tape must be controlled by the DBA, either through operating system commands or by using the recovery manager utility RMAN (described in later chapters).

exam Watch

Archiver processes launch automatically if the database is in archivelog mode.

The transition to archivelog mode can only be done while the database is in mount mode after a clean shutdown, and it must be done by a user with a SYSDBA connection. It is also necessary to set the initialization parameters that control the names and locations of the archive logs generated. Clearly, these names must be unique or archive logs could be overwritten by other archive logs. To ensure unique filenames, it is possible to embed variables such the log switch sequence number in the archive log filenames. These variables may be used to embed unique values in archive log filenames:

Variable	Description
%d	A unique database identifier, necessary if multiple databases are being archived to the same directories.
%t	The thread number, visible as the THREAD# column in V$INSTANCE. This is not significant, except in a RAC database.
%r	The incarnation number. This is important if an incomplete recovery has been done, which will reset the log switch sequence number.
%s	The log switch sequence number. This will guarantee that the archives from any one database incarnation do not overwrite each other.

The minimum archiving necessary to ensure that recovery from a restored backup will be possible is to set one archive destination. But for safety, it will usually be a requirement to multiplex the archive log files by specifying two or more destinations, ideally on different disks served by different controllers. From 9*i* onward, it is possible to specify up to ten archive destinations, giving you ten copies of each filled online redo log file. This is perhaps excessive for safety:

One archive destination? Good idea. Two destinations? Sure, why not. But *ten*?

The reason for ten possible destinations is Data Guard. For the purposes of this book and the OCP exam, an archive log destination will always be a directory on the machine hosting the database—and two destinations on local disks will usually be sufficient. But the destination can be an Oracle Net alias, specifying the address of a listener on a remote computer. This is the key to zero data loss: the redo stream can be shipped across the network to a remote database, where it can be applied to give a real-time backup. Furthermore, the remote database can (if desired) be configured and opened as a data warehouse, meaning that all the query processing can be offloaded from the primary database to a secondary database optimized for such work.

EXERCISE 15-3

Investigate the Redo Log Configuration

In this exercise, you will investigate the configuration of the redo log.

1. Connect to your database as user SYSTEM using SQL*Plus.
2. Document the configuration of redo log with this commands:

   ```
   select * from v$log join v$logfile on v$log.group#=v$logfile
   .group#;
   ```

   This will show the log file members, their status, their size, and the group to which they belong. If your database is the default database, it will have three groups each of one member, 50MB big.
3. Determine the archivelog mode of the database, and whether ARCn is running, with these commands:

   ```
   select log_mode from v$database;
   select archiver from v$instance;
   ```

   Note that the mode is an attribute of the database, but archiving is an attribute of the instance.

4. Connect to your database as user SYSTEM with Database Control, and view the redo log configuration. Take the Server tab on the database home page, and then the Redo Log Groups link in the Storage section.
5. Then in database control, check the archivelog mode. Take the Availability tab on the database home page, and then the Recovery Settings link in the Backup/recovery section. Here you will see the estimated mean time to recovery, and also the archivelog mode.

CERTIFICATION OBJECTIVE 15.04

Overview of Flash Recovery Area

The flash recovery area is a disk destination used as the default location for recovery-related files. It is controlled with two instance parameters:

- db_recovery_file_dest
- db_recovery_file_dest_size

The first of these parameters nominates the location. This can be a file system directory, or an ASM disk group. It is possible for several databases to share a common destination; each database will have its own directory structure (created automatically) in the destination. The second parameter limits the amount of space in the destination that the database will occupy; it says nothing about how much space is actually available in the destination.

The files that will be written to the flash recovery area (unless specified otherwise) include

- Recovery Manager backups
- Archive redo log files
- Database flashback logs

RMAN, the Recovery Manager, can manage space within the flash recovery area: it can delete files that are no longer needed according to its configured policies for retaining copies and backup of files. In an ideal situation, the flash recovery area will be large enough to store a complete copy of the database, plus any archive logs and incremental backups that would be necessary to recover the copy if necessary.

The database backup routines should also include backing up the flash recovery area to tape, thus implementing a strategy of primary, secondary, and tertiary storage:

- Primary storage is the live database, on disk.
- Secondary storage is a copy of the database plus files needed for fast recovery.
- Tertiary storage is long-term backups in a tape library.

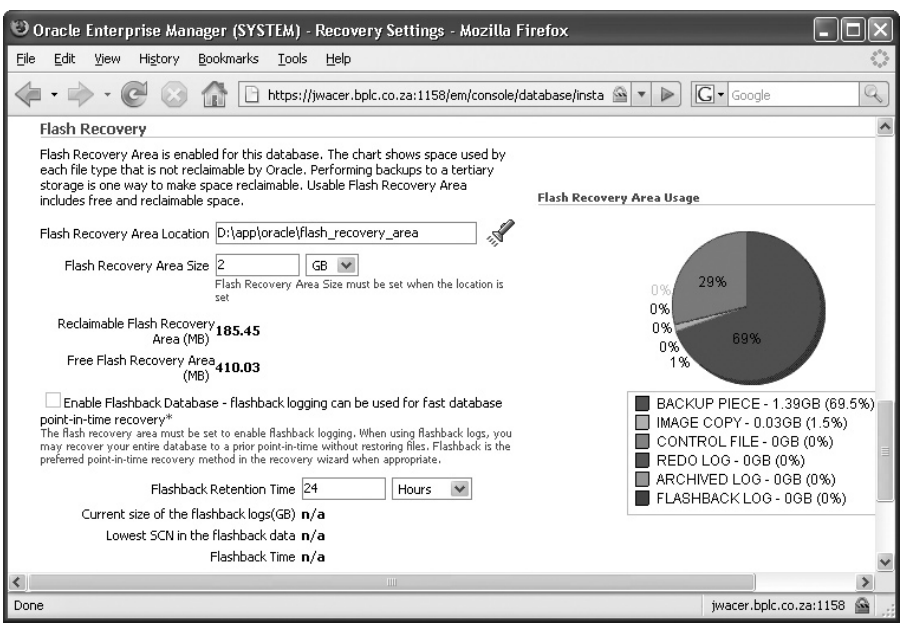
RMAN can manage the whole cycle: backup of the database from primary to secondary and migration of backups from secondary to tertiary storage. Such a system can be implemented in a fashion that will allow near-instant recovery following a failure, combined with the ability to take the database back in time if this is ever necessary.

EXERCISE 15-4

Investigate the Flash Recovery Area Configuration

In this exercise, you will investigate the configuration of the flash recovery area. Using your Database Control session, continue from the last step in Exercise 15-3.

1. Navigate to the bottom part of the Recovery Settings window, which shows the flash recovery area configuration, as shown in this illustration:



2. Interpret the display: note the settings for the instance parameters DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE. These are `D:\app\oracle\flash_recovery_area` and 2 GB in the illustration. Note the space free and the space reclaimable: 410 MB and 185 MB in the illustration. Note the allocation of space: in the illustration, 69 percent of the flash recovery area is taken up with backup pieces, 1.5 percent with image copies.
3. If the flash recovery area parameters are not set, set them now. The directory must be one on which the operating system account that owns the Oracle software has full permissions, and the size should be at least 2 GB.

CERTIFICATION OBJECTIVE 15.05

Configure ARCHIVELOG Mode

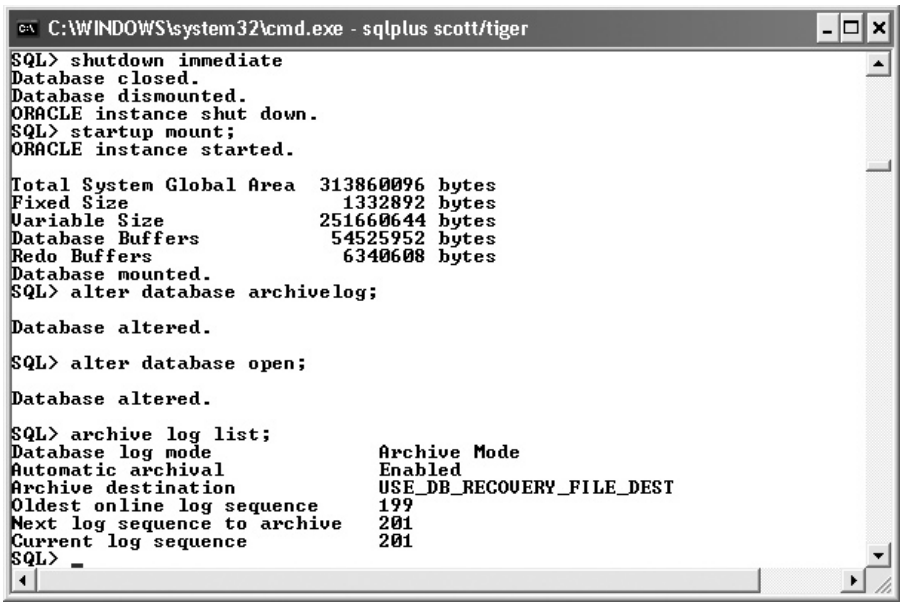
A database is by default created in noarchivelog mode. The transition to archivelog mode is straightforward, but it does require downtime. The process is

- Shut down the database, cleanly.
- Start up in mount mode.
- Issue the command ALTER DATABASE ARCHIVELOG.
- Open the database.
- Perform a full backup.

This sequence can be followed either with SQL*Plus (as shown in Figure 15-5) or with Database Control. Generally, more configuration will be required. Following a default installation, the archive logs will be written to only one destination, which will be the flash recovery area. If the parameters that enable the flash recovery area have not been set, they will go to a platform specific destination (the `$ORACLE_HOME/dbs` directory for Unix systems.) the final command in Figure 15-5, ARCHIVE LOG LIST, shows summary information about the archiving

FIGURE 15-5

Transitioning a database to archivelog mode with SQL*Plus



configuration: that the database is running in archivelog mode; that the ARCn process is running; and that the archive log files are being written to the flash recovery area.

exam Watch

The change to archivelog mode can only be done in mount mode, after a clean shutdown.

Additional changes would usually include adjusting instance parameters to set up a second archive log file destination for multiplexed copies, and defining a naming template for generating archive log file names.

A full backup is an essential step for the transition to archivelog mode. Following the transition, all backups made earlier are useless. The backup can be made while the database is open or closed, but until it is made, the database is not protected at all.

on the Job

If the shutdown is not clean (for instance, SHUTDOWN ABORT), the transition will fail. Not a problem—open the database, and shut it down again: this time, cleanly.

INSIDE THE EXAM

Backup and Recovery Concepts

Backup, restore, recovery, and flashback (in all their forms) are covered in much greater detail in the second DBA OCP examination. At this stage, all that should be required is an understanding of the architecture and the general principles. However, do not skimp on this, because the usual rule applies: if you understand the architecture, you can work out the rest.

So, you must be absolutely clear on the memory structures and the processes involved. The foundation for this is an understanding of the mechanisms of rollback and redo: how they are implemented in the SGA, and in the redo log and the datafiles. Related to this is COMMIT processing.

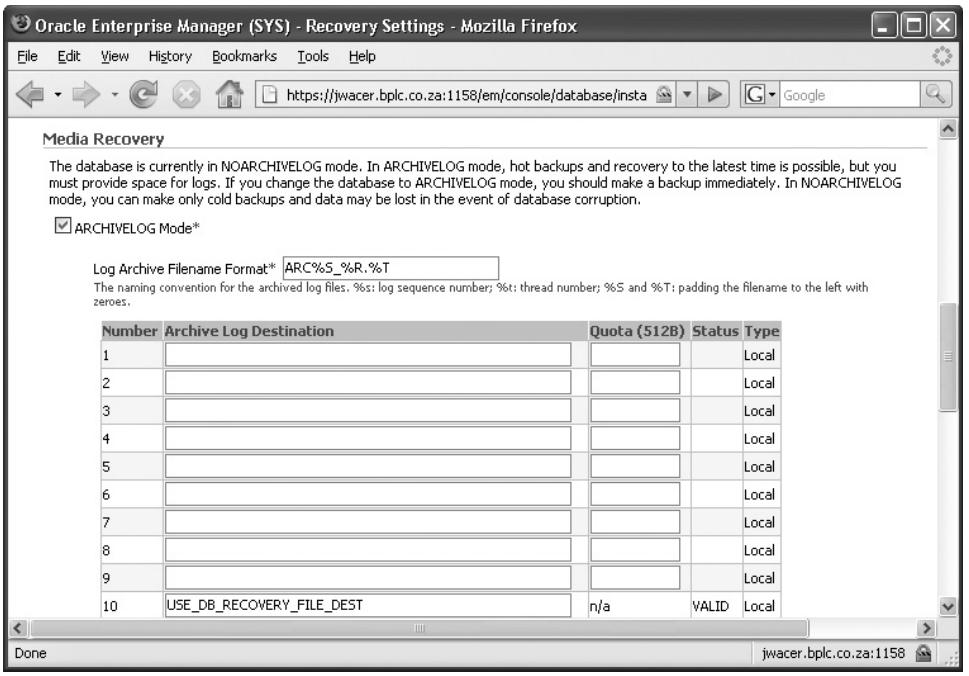
The content of this chapter is not demanding, but it is the foundation on which the next two chapters (and much of the content of the next examination) are built.

EXERCISE 15-5

Enable Archivelog Mode

In this exercise, you will transition your database into archivelog mode. This is an essential procedure for completing the next two chapters on backup and recovery.

1. Connect to your database with Database Control as user SYS. (Why SYS? Because you will have to stop and start the instance.)
2. From the database home page, take the Availability tab and then Recovery Settings link in the Backup/Recovery section.
3. In the Media Recovery section, select the check box ARCHIVELOG Mode, set the Log Archive Filename Format to **ARC%s_%R.%T**, and set the tenth "Archive Log Destination" to USE_DB_RECOVERY_FILE_DEST. This state is shown in this illustration:



4. Click Apply. There will be a prompt to restart the database.
5. After the restart, confirm the mode change with SQL*Plus. Connect with the SYSDBA role and run the ARCHIVE LOG LIST command:

```
connect / as sysdba
archive log list
```

6. Confirm that archiving is working by forcing a log switch and an archive:

```
alter system archive log current;
```

7. Confirm that the archive log file has been generated, in the flash recovery area:

```
select name,is_recovery_dest_file from v$archived_log;
```

CERTIFICATION SUMMARY

There are many types of failure that can occur. Some of them will be repaired automatically (such as instance or session failures); others are not failures at all as far as the database is concerned (such as user errors). There are techniques for repairing user errors, which may need DBA intervention. Where DBA involvement is always needed is in configuring the database for recoverability in the event of media failure. This requires enabling the archivelog mode, and establishing routines for backup. The flash recovery area can simplify the management of all recovery-related files; it is a single storage point, with self-management capabilities for space usage.

TWO-MINUTE DRILL

Identify the Types of Failure That Can Occur in an Oracle Database

- ❑ Instance failure results in automatic instance recovery on the next startup.
- ❑ Session failures are managed automatically by the PMON process.
- ❑ User errors can be reversed using a number of techniques.
- ❑ Recovering from media failures requires use of backups and archive logs.

Describe Ways to Tune Instance Recovery

- ❑ Instance recovery is automatic and unstoppable.
- ❑ Instance recovery applies change vectors from the online redo log files, since the last incremental checkpoint position.
- ❑ The time taken for instance recovery is dependent on the amount of redo to be applied, and the number of I/Os on datafiles needed to apply it.
- ❑ The FAST_START_MTTR_TARGET parameter sets a maximum time for recovery, using a self-tuning mechanism.
- ❑ If FAST_START_MTTR_TARGET is set, it will also enable the checkpoint auto-tuning process to reduce recovery time further.

Identify the Importance of Checkpoints, Redo Log Files, and Archived Log Files

- ❑ Full checkpoints occur only on orderly shutdown, or on demand.
- ❑ Partial checkpoints occur automatically when necessary.
- ❑ Incremental checkpoints advance the point in the redo stream from which recovery must begin after an instance failure.
- ❑ The redo log consists of the disk structures for storing change vectors. The online log is essential for instance recovery.
- ❑ The archive log consists of copies of online log file members, created as they are filled. These are essential for datafile recovery after media failure.

Overview of Flash Recovery Area

- The flash recovery area is configured with the instance parameters DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE.
- The flash recovery area is a default destination for all recovery-related files.

Configure ARCHIVELOG Mode

- In archivelog mode, an online log file member cannot be overwritten until it has been archived.
- Archive processes start automatically when archivelog mode is enabled.
- The mode can only be changed when the database is mounted.

SELF TEST

Identify the Types of Failure That Can Occur in an Oracle Database

1. Different errors require different actions for recovery. Match these types of failure (a) through (d) to the appropriate recovery process A through D:
   (a) Server machine reboots
   (b) Client machine reboots
   (c) Statement causes a constraint violation
   (d) Datafile damaged
   A. PMON
   B. RMAN
   C. Session server process
   D. SMON

Describe Ways to Tune Instance Recovery

2. What instance parameter must be set to enable the checkpoint auto-tuning capability? (Choose the best answer.)
   A. dbwr_io_slaves
   B. fast_start_mttr_target
   C. log_checkpoint_interval
   D. statistics_level
3. Which redo log files may be required for instance recovery? (Choose the best answer.)
   A. Only the current online redo log file
   B. Only the active online redo file(s)
   C. Both current and active online redo log file(s)
   D. Current and active online redo log files, and possibly archive redo log files

Identify the Importance of Checkpoints, Redo Log Files, and Archived Log Files

4. When will a full checkpoint occur? (Choose all correct answers.)
   A. As part of a NORMAL shutdown
   B. As part of an IMMEDIATE shutdown

C. When a tablespace is taken offline
D. When a log switch occurs

5. Which of these operations cannot be accomplished while the database is open? (Choose all correct answers.)
A. Adding a controlfile copy
B. Adding an online log file member
C. Changing the location of the flash recovery area
D. Changing the archivelog mode of the database

6. How can you use checkpointing to improve performance?
A. Frequent checkpoints will reduce the workload on the DBWn.
B. Enabling checkpoint auto-tuning will optimize disk I/O.
C. Reducing the MTTR will reduce disk I/O.
D. Increasing the MTTR will reduce disk I/O.

Overview of Flash Recovery Area

7. What file types will, by default, be stored in the flash recovery area if it has been defined? (Choose all correct answers.)
A. Archive redo log files
B. Background process trace files
C. RMAN backup sets
D. RMAN image copies
E. Undo data

Configure ARCHIVELOG Mode

8. There are several steps involved in transitioning to archivelog mode. Put these in the correct order:
1 alter database archivelog
2 alter database open
3 alter system archive log start
4 full backup
5 shutdown immediate
6 startup mount

A. 5, 6, 1, 2, 4; 3 not necessary
B. 5, 4, 6, 1, 2, 3
C. 6, 1, 3, 5, 4, 2
D. 1, 5, 4, 6, 2; 3 not necessary
E. 5, 6, 1, 2, 3; 4 not necessary

9. What conditions must hold before an online log file member can be reused if the database is operating in archivelog mode? (Choose all correct answers.)

A. It must be inactive.
B. It must be multiplexed.
C. It must be archived.
D. The archive must be multiplexed.

10. If the database is in archivelog mode, what will happen if the archiving fails for any reason? (Choose the best answer.)

A. The instance will abort.
B. All non-SYSDBA sessions will hang.
C. DML operations will hang.
D. The database will revert to noarchivelog mode.

LAB QUESTION

Check whether your database is configured for recoverability. These are the points to look at:

- Online redo log files should be multiplexed.
- The controlfile should be multiplexed.
- The database should be in archivelog mode.
- Archiver processes should be running.
- Archive logs should be multiplexed.
- Checkpoint auto-tuning should be enabled.
- The flash recovery area should be enabled.

SELF TEST ANSWERS

Identify the Types of Failure That Can Occur in an Oracle Database

1. ☑ **A** – b: PMON releases resources of failed sessions. **B** – d: RMAN manages media recovery. **C** – c: the server process rolls back failed statements. **D** – a: SMON performs instance recovery.
   ☒ All other combinations.

Describe Ways to Tune Instance Recovery

2. ☑ **B.** If FAST_START_MTTR_TARGET is set to a non-zero value, then checkpoint auto-tuning will be in effect.
   ☒ **A** is wrong because DBWR_IO_SLAVES is for simulating asynchronous disk I/O. **C** is wrong because LOG_CHECKPOINT_INTERVAL will disable the self-tuning mechanism. **D** is wrong because STATISTICS_LEVEL is not relevant to checkpoint auto-tuning.
3. ☑ **C.** Instance recovery will always require the current online redo log file, and if any others were active, it will need them as well.
   ☒ **A** and **B** are wrong because they are not sufficient. **D** is wrong because instance recovery never needs an archive redo log file.

Identify the Importance of Checkpoints, Redo Log Files, and Archived Log Files

4. ☑ **A** and **B.** Any orderly shutdown will trigger a full checkpoint.
   ☒ **C** is wrong because this would trigger only a partial checkpoint. **D** is wrong because log switches do not trigger checkpoints.
5. ☑ **A** and **D.** Anything to do with the controlfile can only be done in nomount or shutdown modes. Changing the archivelog mode can only be done in mount mode.
   ☒ **B** is wrong because the online redo log can be configured while the database is open. **C** is wrong because DB_RECOVERY_FILE_DEST is a dynamic parameter.
6. ☑ **D.** Setting a longer FAST_START_MTTR_TARGET, or not setting it at all, will reduce the need for DBWn to write dirty buffers to disk, which should improve performance.
   ☒ **A** and **C** are both wrong because they describe the opposite effect. **B** is wrong because the auto-tuning capability does not affect performance, though it will reduce recovery time after instance failure.

Overview of Flash Recovery Area

7. ☑ **A, C,** and **D.** These will go to the flash recovery area, unless directed elsewhere.
☒ **B** is wrong because background trace files will go to a directory in the DIAGNOSTIC_DEST directory. **E** is wrong because undo data is stored in the undo tablespace.

Configure ARCHIVELOG Mode

8. ☑ **A.** This is the correct sequence.
☒ **B, C,** and **D** are wrong because enabling archiving is not necessary (it will occur automatically). **E** is wrong because a backup is a necessary part of the procedure.

9. ☑ **A** and **C.** These are the two conditions.
☒ **B** and **D.** While these are certainly good practice, they are not requirements.

10. ☑ **C.** Once all the online log files need archiving, DML commands will be blocked.
☒ **A** is wrong because the instance will remain open. **B** is wrong because only sessions that attempt DML will hang; those running SELECTs can continue. **D** is wrong because this cannot happen automatically.

LAB ANSWER

The information can be obtained using SQL*Plus, or with Database Control. Using SQL*Plus, these queries would do:

```
select group#,members from v$log;
select count(*) from v$controlfile;
select log_mode from v$database;
select archiver from v$instance;
select count(*) from v$parameter where name like 'log\_archive\_dest\__'
or name like 'log\_archive\_dest\___' escape '\' and value is not null;
select value from v$parameter where name='fast_start_mttr_target';
select value from v$parameter where name='db_recovery_file_dest';
```

If the database is the default database created by DBCA and the exercises have been completed as written, the results of these queries will show that there are two problems: the online redo log file groups are not multiplexed and neither are the archive redo log files.