CHAPTER 6

Writing and Recovery

The Moving Finger Writes; and Having Writ, Moves On
(The Rubaiyat of Omar Khayyam)

In previous chapters you’ve seen how Oracle changes the content of data blocks (including undo blocks) and generates streams of redo information as it does so. You’ve seen the mechanisms that combine changes into transactions and ensure that those transactions appear to be atomic, consistent, and isolated, the first three requirements of ACID, but the questions remains, how does Oracle satisfy the D of ACID and make transactions durable?

So far we’ve been looking at change without making any comments about whether the data we are changing is in memory or on disc. In this chapter we start to take notice of the fact that while most of the work we do starts out in memory, it has to end up on disk at the right time, in the right order, and as efficiently as possible. This means examining the roles of the database writer (dbwr) and the log writer (lgwr), the processes responsible for copying information from memory to disk.

I will also make a few comments about the issue of being able to apply an “infinite” (i.e., arbitrarily large) amount of change and how you avoid the need to create infinitely large files for redo and undo.

Targets

A few years ago I would have said quite happily that there weren’t many databases small enough to fit completely in memory, but that’s no longer true given the very large memories that computers can manage these days. Even so, most of the Oracle databases I see aren’t memory-resident, and even if the database can fit in memory, its owners probably don’t feel safe unless they have a fairly recent snapshot of the data copied to disk. Whatever we do with memory to make our databases go faster, we have to recognize that for most databases the software has to have a good strategy for copying changed memory to disk.

In the case of Oracle we have two types of content to think about, the (buffered) data blocks and the redo log (buffer), and two different but interlinked strategies for writing that content to disk. Of the two sets of content, the redo log buffer is the more important because it is the one that gives us durability with scalability, by minimizing the disk I/O we have to do to secure our database changes. But we need to be sure that the strategies we use for writing both sets of content will always leave us in a position where instantaneous failure of an instance will allow us to restart without losing any committed transactions (which is the durability requirement).
As we have seen, the log file holds a description of all changes we make to the data blocks, so if we can find an old copy of the database somewhere, and all the redo log files that have been written since that copy was made, then we can work through the log files and apply the change records they contain to the database copy to bring the copy up to date.

Through a simple mechanical process we can bring an old copy of the database up to the state it was in just before the instance failed. In particular, if our “old copy” is just a few seconds or a few minutes old because it’s the live production copy of the database that’s on disk and being kept up to date in near real time, then we can recover from instance failure very quickly.

The difficult bit, for Oracle Corp.’s developers, is designing an infrastructure that allows this recovery phase to complete as rapidly as possible while ensuring that any work done in anticipation of recovery is kept to a minimum during normal operation.

The key to being able to recover is writing the redo log to disk in a “timely” fashion; but the key to minimizing recovery time is to keep writing database blocks to disk, and the key to minimizing the work you do writing database blocks is to batch them up and write them in the correct order—which (in the case of Oracle) means oldest first.

So we’re going to look at the activity of the log writer (lgwr) and I’ll explain what I mean by “timely,” and we’ll look at the database writer (dbwr) and I’ll explain the significance of “oldest first” and why that strategy on its own doesn’t quite work.

The problem I have, though, is deciding where to start. Which should come first, lgwr or dbwr? You could argue for the log writer because it must write a change description to disk before the database writer writes the block that has been changed. On the other hand, the database writer sometimes calls the log writer to write, and also generates redo records as it writes data blocks, so there’s a good argument for talking about the database writer first.

Note One of the most important features of the Oracle code is that the database writer will not write a changed block to disk before the log writer has written the redo that describes how the block was changed. This write-ahead logging strategy is critical to the whole recovery mechanism. In effect the redo log files (including the archived ones) form the definitive version of the database; the data files themselves are simply a recent, approximate snapshot. (Any nologging operations invalidate this viewpoint, of course.)

So here we are, going round and round in circles again. I’m going to start with the log writer, then handle the database writer, and then fill in a few of the gaps I left on the first pass through the log writer by describing, in particular, the interactions between the two.

Log Writer

The log writer has one critical task: copy the contents of the log buffer from memory to disk. The activity surrounding this task has, however, increased over the years to cope with increased volumes of change and higher levels of concurrency. (The requirement has also expanded to allow for the possibility of writing to a remote standby database by way of calls to RFS [remote file server] processes from LNS [network server] processes in 10g and NSS or NSA [redo transport sync/async] processes in 11g.

In earlier versions of Oracle there was just a single contiguous area of memory called the log buffer, and, to a very large extent, the appearance of multiple public and private buffers hasn’t changed the