CHAPTER 9

Redo and Undo

…snip….

How Redo and Undo Work Together

In this section, we'll take a look at how redo and undo work together in various scenarios. We will discuss, for example, what happens during the processing of an INSERT with regard to redo and undo generation and how Oracle uses this information in the event of failures at various points in time.

An interesting point to note is that undo information, stored in undo tablespaces or undo segments, is protected by redo as well. In other words, undo data is treated just like table data or index data—changes to undo generates some redo, which is logged. Why this is so will become clear in a moment when we discuss what happens when a system crashes. Undo data is added to the undo segment and is cached in the buffer cache just like any other piece of data would be.

Example INSERT-UPDATE-DELETE Scenario

As an example, we will investigate what might happen with a set of statements like this:

```
insert into t (x,y) values  (1,1);
update t set x = x+1 where x = 1;
delete from t where x = 2;
```

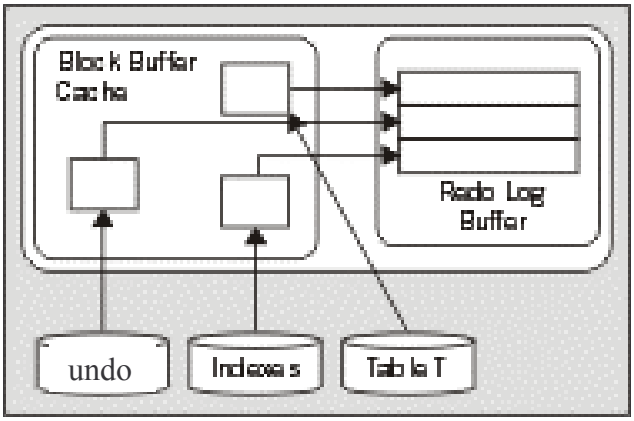
We will follow this transaction down different paths and discover the answers to the following questions:

* What happens if the system fails at various points in the processing of these statements?
* What happens if we ROLLBACK at any point?
* What happens if we succeed and COMMIT?

The INSERT

The initial INSERT INTO T statement will generate both redo and undo. The undo generated will be enough information to make the INSERT "go away." The redo generated by the INSERT INTO T will be enough information to make the insert "happen again."

After the insert has occurred, we have the scenario illustrated in Figure 9-1.



Insert 5300f0901scrap.gif CRX

Figure 9-1. State of the system after an INSERT

There are some cached, modified undo blocks, index blocks, and table data blocks. Each of these blocks is protected by entries in the redo log buffer.

Hypothetical Scenario: The System Crashes Right Now

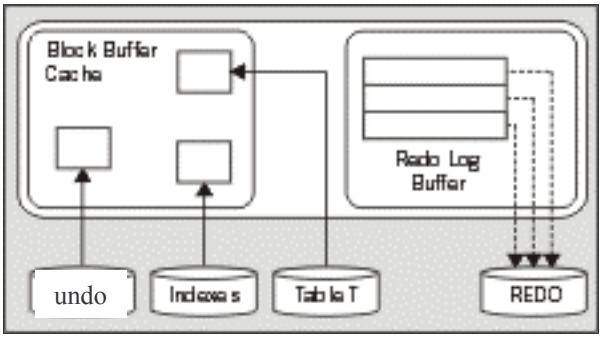
Everything is OK. The SGA is wiped out, but we don't need anything that was in the SGA. It will be as if this transaction never happened when we restart. None of the blocks with changes got flushed to disk, and none of the redo got flushed to disk. We have no need of any of this undo or redo to recover from an instance failure.

Hypothetical Scenario: The Buffer Cache Fills Up Right Now

The situation is such that DBWR must make room and our modified blocks are to be flushed from the cache. In this case, DBWR will start by asking LGWR to flush the redo entries that protect these database blocks. Before DBWR can write any of the blocks that are changed to disk, LGWR must flush the redo information related to these blocks. This makes sense: if we were to flush the modified blocks for table T without flushing the redo entries associated with the undo blocks, and the system failed, we would have a modified table T block with no undo information associated with it. We need to flush the redo log buffers before writing these blocks out so that we can redo all of the changes necessary to get the SGA back into the state it is in right now, so that a rollback can take place.

This second scenario shows some of the foresight that has gone into all of this. The set of conditions described by "If we flushed table T blocks *and* did not flush the redo for the undo blocks *and* the system failed" is starting to get complex. It only gets more complex as we add users, and more objects, and concurrent processing, and so on.

At this point, we have the situation depicted in figure 9-1. We have generated some modified table and index blocks. These have associated undo segment blocks, and all three types of blocks have generated redo to protect them. If you recall from our discussion of the redo log buffer in Chapter 4, it is flushed every three seconds, when it is one-third full or contains 1MB of buffered data, or whenever a commit takes place. It is very possible that at some point during our processing, the redo log buffer will be flushed. In that case, the picture looks like Figure 9-2.

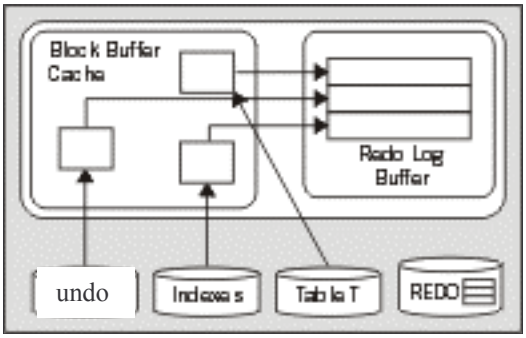


Insert 5300f0902scrap.gif CRX

Figure 9-2. State of the system after a redo log buffer flush

The UPDATE

The UPDATE will cause much of the same work as the INSERT to take place. This time, the amount of undo will be larger; we have some "before" images to save as a result of the update. Now, we have the picture shown in Figure 9-3.



Insert 5300f0903scrap.gif CRX

Figure 9-3. State of the system after the UPDATE

We have more new undo segment blocks in the block buffer cache. To undo the update, if necessary, we have modified database table and index blocks in the cache. We have also generated more redo log buffer entries. Let's assume that some of our generated redo log from the insert is on disk and some is in cache.

Hypothetical Scenario: The System Crashes Right Now

Upon startup, Oracle would read the redo logs and find some redo log entries for our transaction. Given the state in which we left the system, with the redo entries for the insert in the redo log files and the redo for the update still in the buffer, Oracle would "roll forward" the insert. We would end up with a picture much like figure 9-1, with some undo blocks (to undo the insert), modified table blocks (right after the insert), and modified index blocks (right after the insert). Oracle will discover that our transaction never committed and will roll it back since the system is doing crash recovery and, of course, our session is no longer connected. It will take the undo it just rolled forward in the buffer cache and apply it to the

data and index blocks, making them look as they did before the insert took place. Now everything is back the way it was. The blocks that are on disk may or may not reflect the INSERT (it depends on whether or not our blocks got flushed before the crash). If they do, then the insert has been, in effect, undone, and when the blocks are flushed from the buffer cache, the data file will reflect that. If they do not reflect the insert, so be it—they will be overwritten later anyway.

This scenario covers the rudimentary details of a crash recovery. The system performs this as a two-step process. First it rolls forward, bringing the system right to the point of failure, and then it proceeds to roll back everything that had not yet committed. This action will resynchronize the data files. It replays the work that was in progress and undoes anything that has not yet completed.

Hypothetical Scenario: The Application Rolls Back the Transaction

At this point, Oracle will find the undo information for this transaction either in the cached undo segment blocks (most likely) or on disk if they have been flushed (more likely for very large transactions). It will apply the undo information to the data and index blocks in the buffer cache, or if they are no longer in the cache request, they are read from disk into the cache to have the undo applied to them. These blocks will later be flushed to the data files with their original row values restored.

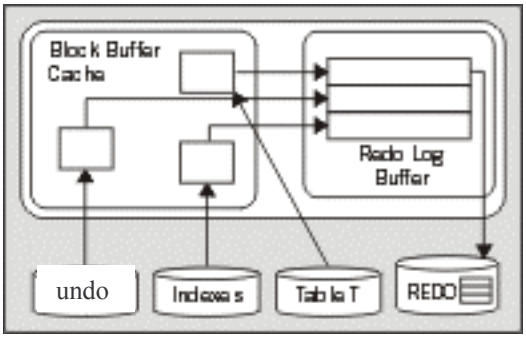
This scenario is much more common than the system crash. It is useful to note that during the rollback process, the redo logs are never involved. The only time redo logs are read is during recovery and archival. This is a key tuning concept: redo logs are written to. Oracle does not read them during normal processing. As long as you have sufficient devices so that when ARCH is reading a file, LGWR is writing to a different device, then there is no contention for redo logs. Many other databases treat the log files as "transaction logs." They do not have this separation of redo and undo. For those systems, the act of rolling back can be disastrous—the rollback process must read the logs their log writer is trying to write to. They introduce contention into the part of the system that can least stand it. Oracle's goal is to make it so that logs are written sequentially, and no one ever reads them while they are being written.

The DELETE

Again, undo is generated as a result of the DELETE, blocks are modified, and redo is sent over to the redo log buffer. This is not very different from before. In fact, it is so similar to the UPDATE that we are going to move right on to the COMMIT.

The COMMIT

We've looked at various failure scenarios and different paths, and now we've finally made it to the COMMIT. Here, Oracle will flush the redo log buffer to disk, and the picture will look like Figure 9-4.



Insert 5300f0904scrap.gif CRX

Figure 9-4. State of the system after a COMMIT

The modified blocks are in the buffer cache; maybe some of them have been flushed to disk. All of the redo necessary to replay this transaction is safely on disk and the changes are now permanent. If we were to read the data directly from the data files, we probably would see the blocks as they existed *before* the transaction took place, as DBWR most likely has not yet written them. That is OK—the redo log files can be used to bring up to date those blocks in the event of a failure. The undo information will hang around until the undo segment wraps around and reuses those blocks. Oracle will use that undo to provide for consistent reads of the affected objects for any session that needs them.

…..