April, 1998

Advisor Answers

Transactions and Buffering

VFP 5.0 and 3.0

Q: I have a BEGIN TRANSACTION/END TRANSACTION sequence in which I use SQL DELETE to remove rows from optimistically table buffered views. When the END TRANSACTION is executed, I expected the data in the views to be committed. Not so. I was able to TableRevert(viewname) and rollback all of the deletes.

Is this the correct behavior for a BEGIN/END TRANSACTION?

- Jeffrey L. Trbovich (via Advisor.Com)

A: In fact, what you saw is correct. Buffering and transactions work together to protect your data's integrity. When you work with views, VFP gives you up to three levels of buffering before your changes are committed to the actual data on disk.

First, views themselves are a kind of buffer. You can work with the data in a view and, until you issue TableUpdate(), the original table is untouched. If you close the view without ever issuing TableUpdate(), the changes are gone.

Second, the table or tables on which a view is based may be buffered as well (although this is optional). If you put the tables on which a view is based in the data environment of a form and the form's BufferMode is something other than 0, the tables are buffered (unless you explicitly set the BufferModeOverride property of the table). When a view is based on a buffered table, issuing TableUpdate() on the view sends the changes from the view to the buffered table. For the changes to be committed to the disk, you also have to issue TableUpdate() for the table.

If the table on which a view is based is not buffered, issuing TableUpdate() for the view does update the table on the disk.

The next level of protection is transactions, which let you ensure that either all the changes to a table go through or none do, and, in fact, that all the changes to all your tables go through together.

When you issue BEGIN TRANSACTION, in essence, VFP adds another level of buffering - call these the "transaction buffers". Within the transaction, any time you issue TableUpdate(), the actual update is sent, not to the tables on disk, but to the transaction buffers. When you issue END TRANSACTION, the transaction buffers are sent to disk all at once. Unless a catastrophe occurs (like a power outage or system crash), all the changes occur - you don't end up with some records updated and others not.

Now, what's going wrong in your situation? The problem is that you're manipulating the view, but you never issue TableUpdate() inside the transaction. Just add the TableUpdate() and all should be well.

However, the approach you're using has a weakness that I should address. In general, it's better to do all your manipulation first, then BEGIN TRANSACTION, issue all your TableUpdate()'s, and END TRANSACTION as quickly as possible. This is because records involved in a transaction are totally locked - others on the network can't access them at all. Obviously, you want to keep them locked for the shortest possible time. There's really no benefit to BEGINning a TRANSACTION before you've finished modifying the data.

There's one other tricky point about transactions. If you end up issuing ROLLBACK (because some table couldn't be updated successfully), be aware that the changes are still sitting in the buffers. To restore the original state of affairs, you need to issue TableRevert() for each of the tables involved. Alternatively, you might analyze the problems that arose and retry the transaction.

The basic structure of the transaction is usually along these lines:

```
* Do all data manipulation
* When done:
BEGIN TRANSACTION
lContinue = TableUpdate("OneTable")
IF lContinue
   lContinue = TableUpdate("AnotherTable")
ENDIF
IF lContinue
   lContinue = TableUpdate("AThirdTable")
ENDIF
* continue for each table involved

* Now commit the updates if successful
IF lContinue
   END TRANSACTION
ELSE
   ROLLBACK
   * At this point there are a couple of choices
   * Either TableRevert() each table involved
   * or try some conflict resolution
ENDIF
```

VFP's buffering and transactions give us more control than ever before over the process of committing data to disk. Using these tools, we can prevent inconsistent data from getting into our databases, except when the truly unusual happens.

- Tamar