EOF-Using EOF-Optimistic Locking

Overview

Note that this section covers the locking strategy EOF uses at the databases layer as opposed to EOEditingContext locking which happens much higher in the stack. For information on EOEditingContext locking, please see Context and Database Locking.

There are two primary styles of locking that can be used with a database API: pessimistic and optimisic. Pessimistic locking actively prevents two participants from updating the same row by explicitly locking access to it when the first participant begins to make a change (effectively locking others out until the change is complete). Optimistic locking allows participants to make changes to the resources without an explicit lock, assuming success, and fails only if the changes actually cause a conflict. EOF takes the optimistic approach when locking data.

As an example of how optimistic locking in EOF works, assume you have a Person EO with the PersonID 123, first name "Bob", and last name "Roberts". If you change the first name to "Robert" and saveChanges() on your editing context, you would see the following SQL sent to your database:

```
UPDATE Person SET FirstName = 'Robert' WHERE PersonID = 123 AND FirstName = 'Bob' AND LastName = 'Roberts'
```

The key to optimistic locking is the where clause on the update. If you were the only one to change this EO, this update command would return that 1 row was updated, which EOF expected as a successful result. However, assume another user was editing at the same time and changed Bob's last name to 'Wilson'. Rerun the above query, and you'd find that ZERO rows would be updated, because the LastName = 'Roberts' clause in the qualifier would not match (the last name in the database is now 'Wilson'). The result of this situation in EOF would be an optimistic locking exception thrown from saveChanges().

When you run EOModeler, you may have noticed the column in the Attributes view with the lock icon in its heading. This column defines which attributes of your entity will be used for optimistic locking. All attributes that are checked will appear in the 'where' clause of an update to your object. Note that all locked columns are checked on an update, not just the columns of the attributes that were modified.

A common problem people run into with EOF is the use of "imprecise" types as locking columns. By imprecise types, I am referring to types whose precision may change during the lifetime of an EO even when it has not been modified by a user. The most notorious data types for this problem are dates and floating point types. It is very common that the precision of Java for floating point types does not match the precision of your database. The result of this is that when EOF brings the float out of the database, puts it in a Float, and places it in your snapshost, the very act of loading it into a Float changes its value. This can often cause very strange problems with locking where you will end up with optimistic locking failures even when there was no concurrent update. The recommendation for this case is to uncheck the Lock column on attributes with date and floating point types if your database is causing problems. The obvious downside of this is that these fields will not be elligible for detection of optimistic locking failures (that is, "last write wins" for that column and your program won't be notified of a collision of updates).

mmalcolm gave a presentation at O'Reilly's 2002 Mac OS X conference titled "A Lack of Conflicts in EOF, or 'Hey Mom, Someone Overwrote My Data!'" and the slides are available at http://conferences.oreillynet.com/presentations/macosx02/crawford_eoconflicts.pdf

Optimistic Locking and Performance

There have been several threads of discussion about the additional database server load with optimistic locking (as the database much verify all of the locked column values prior to an update).

In January 2006, a benchmark was run on WebObjects 5.3 with the following scenario:

This was run on the latest version of FrontBase as of Jan 2006 on an Intel iMac with WO 5.3:

Two identical entities were created which have 3 ints, 3 booleans, 3 strings, and 3 dates as properties, but one locks all attrs and one locks none. 10,000 instances of each were inserted into the db, followed by an update of the same attribute for each with the same change in value.

For the entity with all columns locked, it took 25753ms to complete the run. For the entity with no columns locked, it took 7111ms to complete the run. It was 3.6x faster to not lock any columns in this example, with this database, on this hardware.

Modifying the scenario to lock on one date column results in the entity with the lock taking 25574ms to complete the run, while the unlocked entity took 14125ms to complete, for a speed difference of 1.81x.

It really depends on the type of application you have. If you're doing 100,000 updates, you probably care that it takes an extra 190 seconds (or whatever the time differential might be) to perform the update. But if you're just periodically updating "normal" datasets, it may not matter that it takes an extra few milliseconds. The trade-off, obviously, is data integrity.

Database benchmarks are never simple, and often don't apply across databases of different vendors, so take these results with a grain of salt. Always profile your application before you make large sacrifices in functionality.

Seeing it Occur

Kieran Kelleher

Here is the easiest way to trigger it.

1. Pull an object into an edit page in WO.

2. Now go in the back door to the database using command line or some other tool independent of WebObjects and directly edit some attributes (that have the little lock ON in EOModel) of the databases record that represents the object.
3. Now submit your object editing page
4. An optimistic locking exception will now get thrown because the original snapshot that was read in from the database before you edited the object is different to the values in the database now.

Let's say we have a table with a primary key 'oid' and one optimistically locked attribute named 'field1'.
Now we fetch the object with primary key 273 and its DB value is "original".

Now let's say you change the field1 to be "new" in WO in a form and BEFORE you submit, let's say you changed the DB field1 value directly to "changed", then when you submit, EOF generates SQL something like:
UPDATE table T1 set field1 = "new" where oid = 273 and field1 = "original";

.... and this update will FAIL because there is NO record having oid = 273 and field1 = "original". There IS a record with oid = 273 and field1 = "changed", but WO does not know that because some other application has changed it.

If you log out the SQL while changing you will see that optimistic locking effects the WHERE part of all the UPDATE statements that are generated.