Introduction to Multi-Statement Transactions

June 21, 2013 at 12:34 PM | categories: XQuery, MarkLogic | 0 Comments

If you are an old hand with MarkLogic, you are used to writing update queries with implicit commits. Sometimes this means restructuring your code so that everything can happen in one commit, with no conflicting updates. In extreme cases you might even decide to run multiple transactions from one query, using `xdmp:invoke` or semicolons. Historically this meant giving up atomicity.

Multi-statement transactions, introduced in MarkLogic 6, promise a third way. We can write a transaction that spans multiple statements, with an explicit commit or rollback.

For most updates it's probably best to stick with the old ways and use implicit commits. But let's look at a concrete example of a time when multi-statement transactions are the right tool for the job.

Suppose you are using DLS (Document Library Services) to manage your document versioning. But you have a special case where you want to insert two discrete versions of the same document atomically. That may sound odd, but I ran into that exact problem recently.

First we need to discover that there is a problem. Let's bootstrap the a test document with DLS.

```
import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";
try {
  dls:document-delete('test', false(), false()) }
catch ($ex) {
  if ($ex/error:code ne 'DLS-UNMANAGED') then xdmp:rethrow()
  else if (empty(doc('test'))) then ()
  else xdmp:document-delete('test') }
;
import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";
dls:document-insert-and-manage('test', false(), <x id="x1"/>)
```

Now let's write some XQuery to insert two versions in one update, and see what happens.

```
import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";
dls:document-checkout-update-checkin(
  'test', <x id="x2"/>, "version two", true()),
dls:document-checkout-update-checkin(
  'test', <x id="x3"/>, "version three", true())
```

This throws an `XDMP-CONFLICTINGUPDATES` error, because these calls to DLS end up trying to update the same nodes twice in the same transaction. In implicit commit mode, aka "auto" mode, this is difficult to avoid. We could ask MarkLogic to extend DLS with a new function designed for this situation. But that is a long-term solution, and we need to move on with this implementation.

So what can we do? We might read up on `xdmp:invoke`, `xdmp:eval`, etc. If we are careful, we can write a top-level read-only query that invokes one or more update transactions.

```
(: Entry point - must be a read-only query. :)
xdmp:invoke(
  'update.xqy',
```

```
  (xs:QName('URI'), 'test',
   xs:QName('NEW'), <x id="x2"/>,
   xs:QName('NOTE'), "version two")),
xdmp:invoke(
  'update.xqy',
  (xs:QName('URI'), 'test',
   xs:QName('NEW'), <x id="x3"/>,
   xs:QName('NOTE'), "version three"))
```

This invokes a module called `update.xqy`, which would look like this:

```
(: update.xqy :)
import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";

declare variable $NEW as node() external ;
declare variable $NOTE as xs:string external ;
declare variable $URI as xs:string external ;

dls:document-checkout-update-checkin(
  $URI, $NEW, $NOTE, true())
```

This works - at least, it doesn't throw `XDMP-CONFLICTINGUPDATES`. But we have lost atomicity. Each of the two updates runs as a different transaction. This opens up a potential race condition, where a second query updates the document in between our two transactions. That could break our application.

There are ways around this, but they get complicated quickly. They are also difficult to test, so we can never be confident that we have plugged all the potential holes in our process. It would be much more convenient if we could run multiple statements inside one transaction, with each statement able to see the database state of the previous statements.

We can do exactly that using a multi-statement transaction. Let's get our feet wet by looking at a very simple MST.

```
declare option xdmp:transaction-mode "update";

xdmp:document-insert('temp', <one/>)
;

xdmp:document-insert('temp', <two/>),
xdmp:commit()
```

There are three important points to this query. 1. The option`xdmp:transaction-mode="update"` begins a multi-statment transaction. 1. The semicolon after the first `xdmp:document-insert` ends that statement and begins another. 1. The `xdmp:commit`ends the multi-statement transaction by commiting all updates to the database.

This runs without error, and we can verify that `doc('temp')` contains `<two/>` after it runs. But how can we prove that all this takes place in a single transaction? Let's decorate the query with a few more function calls.

```
declare option xdmp:transaction-mode "update";

xdmp:get-transaction-mode(),
xdmp:transaction(),
doc('temp')/*,
```

```
xdmp:document-insert('temp', <one/>)
;

xdmp:get-transaction-mode(),
xdmp:transaction(),
doc('temp')/*,
xdmp:document-insert('temp', <two/>),
xdmp:commit()
```

This time we return some extra information within each statement: the transaction mode, the transaction id, and the contents of the test doc. The transaction ids will be different every time, but here is one example.

```
update
17378667561611037626
<two/>
update
17378667561611037626
<one/>
```

So the document `test` started out with the old node `<two/>',
but after the first statement it changed to`. Both statements see the same transaction mode and id.

Try changing the `xdmp:transaction-mode` declaration to `auto`, the default. You should see the mode change to `auto`, and two different transaction-ids. This tells us that in `update` mode we have a multi-statement transaction, and in `auto` mode we have a non-atomic sequence of two different transactions. Before MarkLogic 6, all update statements ran in `auto` mode.

Now let's apply what we've learned about MST to the original problem: inserting two different versions of a managed document in a single transaction.

```
import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";

declare option xdmp:transaction-mode "update";

dls:document-checkout-update-checkin(
  'test', <x id="x2"/>, "version two", true())
;

import module namespace dls="http://marklogic.com/xdmp/dls"
  at "/MarkLogic/dls.xqy";
dls:document-checkout-update-checkin(
  'test', <x id="x3"/>, "version three", true()),
xdmp:commit()
```

As above, this code uses three important features: 1. Set `xdmp:transaction-mode="update"` to begin the MST. 1. Use semicolons to end one statement and begin another. 1. Use`xdmp:commit` to end the MST and commit all updates.

To abort a multi-statement transaction, use `xdmp:rollback`.

So now you have a new tool for situations where implicit commit is a little too awkward. Try not to overdo it, though. In most situations, the default `xdmp:transaction-mode="auto"` is still the best path.