Database - Bug #4035

newly created record is not flushed at the right time

04/08/2019 05:09 PM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eric Faulhaber	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Database - Bug #2222: WRITE event triggered too early	**Hold**
Related to Base Language - Bug #4149: BlockManager and TransactionManager han...	**New**

History

#1 - 04/08/2019 05:10 PM - Eric Faulhaber

- Related to Bug #2222: WRITE event triggered too early added

#2 - 04/08/2019 05:27 PM - Eric Faulhaber

The following test case illustrates the problem.

Constantin Asofiei wrote:

> Good news, I have a recreate; I think the condition is that the second record is not fully assigned - its indexed fields are not set. See these two programs:
>
> - buf1.p - yes, the nested for each tt blocks are required, otherwise buf2.p will not find the second record - even in 4GL.

```
def temp-table tt1 field f1 as int.
def temp-table tt2 field f1 as int.

create tt1.
tt1.f1 = 1.
create tt2.
tt2.f1 = 1.

for each book:
   delete book.
end.

def var i as int.
do transaction:

   for each tt2 transaction:
    for each tt1 transaction:
       create book.
       assign book.book-id = 1
              book.isbn = "abc"
              book.book-title = "a".
    end.

    create book.
    book.book-title = "b".
   end.

   run buf2.p.
end.
```

```
for each book:
   display book.
end.
```

- buf2.p

```
for each book by book.book-title:
   display book.book-id book.isbn book.book-title with frame f1 4 down.
end.
```

In 4GL, buf2.p finds both records. In FWD, just the first one.

The following patch fixed the issue in this test case:

```
=== modified file 'src/com/goldencode/p2j/persist/RecordBuffer.java'
--- src/com/goldencode/p2j/persist/RecordBuffer.java    2019-03-15 09:24:46 +0000
+++ src/com/goldencode/p2j/persist/RecordBuffer.java    2019-03-17 20:54:00 +0000
@@ -1017,6 +1017,8 @@
 **                            copy of a record with lazy extent fields.
 **     ECF 20190314           Ensure we are deleting the correct DMO instance when rolling back a
 **                            ReversibleCreate.
+**     ECF 20190317           Provisionally modified validate(boolean) to flush unconditionally for
+**                            undoable buffers.
 */

 /*
@@ -4814,6 +4816,11 @@
       boolean flush = false;
       boolean trigger = false;

+      /* TODO: the following logic seems very specifically limited, but the combined test cases
+       * flush-buf1.p and flush-buf2.p show that a write trigger is fired and the second record
+       * is written to the database after the outer subtransaction commits. This change fixes
+       * that test case scenario, but I'm not sure it's fully correct. Thus, I'm leaving this
+       * commented code here for now. ECF
       boolean global = registeredWithGlobalScope == 0;
       boolean exitingActiveScope = blockDepth == activeScopeDepth && (!global || !worldScope);

@@ -4828,7 +4835,9 @@
          flush = isTransient();
          trigger = true;
       }
-      else if (!isUndoable())
+      else*/
+
+      if (!isUndoable())
       {
          // no-undo buffer that is not in a transaction, but has a dirty, transient record must
          // try to flush it (which will trigger validation)
@@ -4836,7 +4845,9 @@
       }
       else
       {
-         return;
+         flush = isTransient();
+         trigger = true;
+//         return;  // see comments above
       }

       if (trigger && isAvailable())
```

For reference/context purposes, this patch was made in rev 11498 of task branch 3750b.

As noted in the comments, I was concerned that this might cause regressions. It did in fact regress real applications, by flushing certain transient records too aggressively. This resulted in all manner of downstream problems, generally manifesting as various forms of "record not available" errors or Hibernate LazyInitializationException. However, because of the nature of the root cause, it could manifest in other ways as well.

So far, any time I've fixed either the above test case, or the applications, the fix for one regresses the other. We need to derive some test cases from the regressed applications which, in combination with the above test case, indicate the true behavior of the 4GL. The test cases we used for #2222 might be of use in this regard. Some effort is needed to get these in working order again.

#3 - 07/11/2019 02:53 PM - Eric Faulhaber

Here are the compile listings for the test cases above (note: I have renamed them to flush-buf1.p and flush-buf2.p):

flush-buf1.p:

```
{} Line Blk
-- ---- ---
      1     def temp-table tt1 field f1 as int.
      2     def temp-table tt2 field f1 as int.
      3
      4     create tt1.
      5     tt1.f1 = 1.
      6     create tt2.
      7     tt2.f1 = 1.
      8
      9   1 for each book:
     10   1    delete book.
     11     end.
     12
     13     def var i as int.
     14   1 do transaction:
     15   1
     16   2    for each tt2 transaction:
     17   3     for each tt1 transaction:
     18   3        create book.
     19   3        assign book.book-id = 1
     20   3               book.isbn = "abc"
     21   3               book.book-title = "a".
     22   2     end.
     23   2
     24   2     create book.
     25   2     book.book-title = "b".
     26   1    end.
     27   1
     28   1    run flush-buf2.p.
     29     end.
     30
     31   1 for each book:
     32   1    display book.
     33     end.
```

```
     File Name       Line Blk. Type   Tran            Blk. Label
-------------------- ---- ----------- ---- --------------------------------
e:\tc\flush-buf1.p      0 Procedure   No
    Buffers: book.Book
             tt2
             tt1

e:\tc\flush-buf1.p      9 For         Yes
e:\tc\flush-buf1.p     14 Do          Yes
e:\tc\flush-buf1.p     16 For         Yes
e:\tc\flush-buf1.p     17 For         Yes
e:\tc\flush-buf1.p     31 For         No
    Frames:  Unnamed
```

flush-buf2.p:

```
{} Line Blk
-- ---- ---
      1   1 for each book by book.book-title:
      2        display book.book-id book.isbn book.book-title with frame f1 4 do
      2   1 wn.
      3     end.

     File Name       Line Blk. Type   Tran            Blk. Label
-------------------- ---- ----------- ---- --------------------------------
e:\tc\flush-buf2.p      0 Procedure   No
e:\tc\flush-buf2.p      1 For         No
    Buffers: book.Book
    Frames:  f1
```

#4 - 07/11/2019 02:56 PM - Greg Shah

Are the buffer scopes the same as the converted code? Based on what you told me previously, the transaction scoping should be the same.

#5 - 07/11/2019 03:05 PM - Eric Faulhaber

Greg Shah wrote:

> Are the buffer scopes the same as the converted code? Based on what you told me previously, the transaction scoping should be the same.

Yes, in FlushBuf1.java, we've got Transaction.FULL being passed into the delete loop forEach, the doBlock, and both levels of the nested forEach blocks, which corresponds with the listing:

```
public class FlushBuf1
{
   Tt2_1_1.Buf tt2 = TemporaryBuffer.define(Tt2_1_1.Buf.class, "tt2", "tt2", false);

   Tt1_1_1.Buf tt1 = TemporaryBuffer.define(Tt1_1_1.Buf.class, "tt1", "tt1", false);

   Book.Buf book = RecordBuffer.define(Book.Buf.class, "p2j_test", "book", "book");

   FlushBuf1Frame0 frame0 = GenericFrame.createFrame(FlushBuf1Frame0.class, "");

   /**
    * External procedure (converted to Java from the 4GL source code
    * in flush-buf1.p).
    */
   public void execute()
   {
      externalProcedure(FlushBuf1.this, new Block((Body) () -> 
      {
         RecordBuffer.openScope(tt1, tt2, book);
         tt1.create();
         tt1.setF1(new integer(1));
         tt2.create();
         tt2.setF1(new integer(1));

         PreselectQuery query0 = new PreselectQuery();

         forEach(TransactionType.FULL, "loopLabel0", new Block((Init) () -> 
         {
            query0.initialize(book, ((String) null), null, "book.bookId asc");
         }, 
         (Body) () ->
         {
            query0.next();
            book.deleteRecord(() -> isNotEqual(book.getBookTitle(), "Bogus Programming"), "
Bogus Programming may not be deleted!!!");
         }));

         doBlock(TransactionType.FULL, "blockLabel0", new Block((Body) () ->
         {
            AdaptiveQuery query1 = new AdaptiveQuery();

            forEach(TransactionType.FULL, "loopLabel1", new Block((Init) () ->
            {
               query1.initialize(tt2, ((String) null), null, "tt2.id asc");
            },
            (Body) () ->
            {
               query1.next();

               AdaptiveQuery query2 = new AdaptiveQuery();

               forEach(TransactionType.FULL, "loopLabel2", new Block((Init) () ->
               {
                  query2.initialize(tt1, ((String) null), null, "tt1.id asc");
               },
               (Body) () ->
               {
                  query2.next();
                  book.create();
                  batch(() ->
                  {
                     book.setBookId(new integer(1));
                     book.setIsbn(new character("abc"));
                     book.setBookTitle(new character("a"));
                  });
               }));

               book.create();
               book.setBookTitle(new character("b"));
            }));

            ControlFlowOps.invoke("flush-buf2.p");
         }));

         AdaptiveQuery query3 = new AdaptiveQuery();

         forEach("loopLabel3", new Block((Init) () ->
         {
            frame0.openScope();
            query3.initialize(book, ((String) null), null, "book.bookId asc");
         },
         (Body) () ->
         {
            query3.next();

            FrameElement[] elementList0 = new FrameElement[]
            {
               new Element(new FieldReference(book, "bookId"), frame0.widgetBookId()),
               new Element(new FieldReference(book, "bookTitle"), frame0.widgetBookTitle()),
               new Element(new FieldReference(book, "publisher"), frame0.widgetPublisher()),
               new Element(new FieldReference(book, "isbn"), frame0.widgetIsbn()),
               new Element(new FieldReference(book, "onHandQty"), frame0.widgetOnHandQty()),
               new Element(new FieldReference(book, "cost"), frame0.widgetCost()),
               new Element(new FieldReference(book, "pubDate"), frame0.widgetPubDate()),
               new Element(new FieldReference(book, "authorId"), frame0.widgetAuthorId()),
               new Element(new FieldReference(book, "soldQty"), frame0.widgetSoldQty()),
               new Element(new FieldReference(book, "price"), frame0.widgetPrice())
            };

            frame0.display(elementList0);
         }));
      }));
   }
}
```

FlushBuf2.java has no Transaction flags, which also matches its listing:

```
public class FlushBuf2
{
   Book.Buf book = RecordBuffer.define(Book.Buf.class, "p2j_test", "book", "book");

   FlushBuf2F1 f1Frame = GenericFrame.createFrame(FlushBuf2F1.class, "f1");

   /**
    * External procedure (converted to Java from the 4GL source code
    * in flush-buf2.p).
    */
   public void execute()
   {
      externalProcedure(FlushBuf2.this, new Block((Body) () -> 
      {
         PreselectQuery query0 = new PreselectQuery();

         forEach("loopLabel0", new Block((Init) () -> 
         {
            f1Frame.openScope();
            RecordBuffer.openScope(book);
            query0.initialize(book, ((String) null), null, "book.bookTitle asc, book.bookId asc");
         }, 
         (Body) () -> 
         {
            query0.next();

            FrameElement[] elementList0 = new FrameElement[]
            {
               new Element(new FieldReference(book, "bookId"), f1Frame.widgetBookId()),
               new Element(new FieldReference(book, "isbn"), f1Frame.widgetIsbn()),
               new Element(new FieldReference(book, "bookTitle"), f1Frame.widgetBookTitle())
            };

            f1Frame.display(elementList0);
         }));
      }));
   }
}
```

#6 - 07/11/2019 03:10 PM - Constantin Asofiei

Eric, some thoughts: for indexed fields, maybe we should assume that when a (sub)tx block iterates, its indexed fields are forced to be assigned to the default values, when no explicit assign was made to them? As I recall, the issue is that we are not assigning anything to the indexed fields - if we assign something, then FWD behaves correctly.

#7 - 07/11/2019 03:17 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric, some thoughts: for indexed fields, maybe we should assume that when a (sub)tx block iterates, its indexed fields are forced to be assigned to the default values, when no explicit assign was made to them? As I recall, the issue is that we are not assigning anything to the indexed fields - if we assign something, then FWD behaves correctly.

That is a great idea. I will investigate this. Thanks!

#8 - 07/11/2019 03:20 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin Asofiei wrote:
>
>> Eric, some thoughts: for indexed fields, maybe we should assume that when a (sub)tx block iterates, its indexed fields are forced to be assigned to the default values, when no explicit assign was made to them? As I recall, the issue is that we are not assigning anything to the indexed fields - if we assign something, then FWD behaves correctly.
>
> That is a great idea. I will investigate this. Thanks!

Just to be sure I'm clear: first change the 4GL program so that the second create book. assigns all indexed fields, and run in FWD - if it behaves correctly, then this theory is confirmed.

#9 - 07/11/2019 03:45 PM - Eric Faulhaber

Yes, it works if ALL indexed fields are updated after the second create book.

However, I'm not sure that forcing all otherwise un-updated, indexed fields to their defaults on a sub-transaction (or full transaction) boundary is the right solution, because isn't that essentially what my previous change in RecordBuffer.validate(boolean) was doing? If a (sub-)transaction boundary forces all indexed fields to their defaults, doesn't that just make it impossible to return from that method without flushing? That's essentially what my previous change did, but it caused major regressions.

#10 - 07/11/2019 03:53 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Yes, it works if ALL indexed fields are updated after the second create book.

Even if you explicitly assign them to their default value, right?

> If a (sub-)transaction boundary forces all indexed fields to their defaults, doesn't that just make it impossible to return from that method without flushing? That's essentially what my previous change did, but it caused major regressions.

What I meant here is this: when an indexed field was never assigned in the current iteration for a newly created record, we need to ensure that FWD has processed this field, as if it was explicitly assigned to its default value and RecordBuffer$Handler.invoke was executed. What I think we are missing is all the processing which is performed when an indexed field is assigned - that code needs to be either forced to be executed (by assigning the field to its default value) or identify and execute the missing parts, when the block iterates.

#11 - 07/14/2019 10:11 PM - Eric Faulhaber

Greg, aggressive buffer flush mode support was added in 4035a/11327. It fixes the buf1.p/buf2.p test case and fixes the regression in the first application case I retested. Please review.

I have not yet codified all the test case permutations we discussed.

#12 - 07/14/2019 10:44 PM - Greg Shah

Code Review Task Branch 4035a Revision 11327

I'm OK with the changes. I think they accurately implement the behavior.

#13 - 07/14/2019 11:00 PM - Greg Shah

To summarize the idea:

- Any FOR BLOCK or FOR LOOP causes a special aggressive flushing mode to occur when all the following conditions are met:
    - Changes have been made to a buffer, which can be as simple as a CREATE or the assignment of at least one field. It does not matter if the field is part of an index or not.
    - The changes must be made in code that is in the same external procedure file as the outermost FOR BLOCK/FOR LOOP.
    - That code can be directly in the FOR BLOCK/FOR LOOP but it can also be in any nested inner block, internal procedure, function and presumably triggers though we haven't checked that.
    - The next sub-transaction exit that occurs within that FOR BLOCK/FOR LOOP scope but after the changes, will flush the pending changes

    to the database. This assumes that the full transaction level is higher than the block with the changes, since if they are the same block then it would be the full transaction block and the changes would be flushed anyway. So this extra flushing kicks in only for sub-transactions.
    - Blocks without sub-transaction property are ignored.
    - Other block types such as DO PRESELECT/REPEAT PRESELECT do not have this behavior.
    - This special mode is suspended when an external procedure is executed. But any sub-transaction that occurs after the external procedure is executed but still inside a FOR BLOCK/FOR LOOP, will still cause this extra flushing.
    - Other than external procedures, the block type of the nested code doesn't seem to matter so long as it has sub-transaction property.
- An external procedure clears the FOR BLOCK/FOR LOOP flushing mode for as long as that external proc is executing, BUT if there are any pending changes made during the external procedure's execution, they will be flushed when the external procedure exits. This is different from the FOR BLOCK/FOR LOOP behavior because the nested sub-transactions of an external procedure do not have the extra flushing (unless enclosed in a FOR BLOCK/FOR LOOP in that external proc). Those changes just build up until the external procedure exits and then the extra flushing will occur.

This was proven by a large number of manually implemented tests. Eric will be organizing them into a proper set of tests to prove the various cases. I think he will need 20 or so tests to so the variations.

#14 - 07/15/2019 04:13 PM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from New to Closed

4035a/11327 passed main regression testing and ETF. I rebased to trunk rev 11325, merged to trunk, and committed as trunk rev 11326. I archived branch 4035a.

#15 - 07/16/2019 06:32 PM - Greg Shah

- Related to Bug #4149: BlockManager and TransactionManager handle TransactionType.NONE incorrectly added