Database - Bug #2222

WRITE event triggered too early

01/08/2014 12:41 PM - Ovidiu Maxiniuc

Status:	Hold	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eric Faulhaber	**% Done:**	80%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #2024: implement runtime support for database t...	**Closed**	**06/19/2013**	**07/01/2013**
Related to Database - Support #2371: trigger behavior	**Closed**		
Related to Database - Bug #4035: newly created record is not flushed at the r...	**Closed**		

History

#1 - 01/08/2014 02:08 PM - Ovidiu Maxiniuc

- File write-trigger-test5.p.lst added

- File write-trigger-test5.p added

In some conditions, the WRITE event caused by the flush of the record to database occurs earlier than the corresponding event from P4GL.
Attached are a sample write-trigger-test5.p that proves the issue and the listing resulted from compile operation on it.
The listing shows that the book buffer scope is the whole external procedure.
Probably the easiest way to observe the event triggering is using the following table:

P2J	Progress 4GL	Observations
1 Outer transaction start	1 Outer transaction start	
2 Inside procedure start	2 Inside procedure start	
3 Inner transaction start	3 Inner transaction start	
4 Session trigger CREATE for	4 Session trigger CREATE for	
5 Inner transaction end	5 Inner transaction end	
6 Session trigger WRITE for old		a) This WRITE event occurs in P2J too early, when the transaction block from inside the procedure ends.
7 Inside procedure end	6 Inside procedure end	
8 Outer transaction continue	7 Outer transaction continue	
9 Session trigger ASSIGN for 111 old	8 Session trigger ASSIGN for 111 old	
	9 Session trigger WRITE for 111 old	b) This should be the correct moment when P4GL generates

		the WRITE event (buffer is needed to store new record of CREATE).
10 Session trigger CREATE for	10 Session trigger CREATE for	
11 Session trigger ASSIGN for 222 old	11 Session trigger ASSIGN for 222 old	
12 Session trigger WRITE for 222 old		c) Again, WRITE is too early in P2J.
13 Outer transaction end	12 Outer transaction end	
14 Top scope continue	13 Top scope continue	
	14 Session trigger WRITE for 222 old	d) Correct moment when the WRITE from c) should have been emitted (buffer is needed to store data retrieved from db).
15 Session trigger FIND for 222	15 Session trigger FIND for 222	
16 Session trigger DELETE for 222	16 Session trigger DELETE for 222	
17 Session trigger FIND for 111	17 Session trigger FIND for 111	
18 Session trigger DELETE for 111	18 Session trigger DELETE for 111	
19 Top scope end	19 Top scope end	

Much of the lines form output is not relevant, but uses to locate the points where the WRITE events are emitted.

From the above lines it is obvious that P4GL does not care too much about the transactions blocks.

- The 1st WRITE is emitted very late, leaving 2 blocks end their scope (the DO TRANSACTION, at line 17 and the procedure executed at line 28). Instead the buffer is flushed to database only when the space is needed for storing new data, ie. just before the CREATE of the external DO block.
- The 2nd WRITE is also not emitted when the logical block ends (line 26, be it simple DO or DO TRANSACTION, there is no difference) and the data is not written to db until the buffer is needed for storing the data retrieved by the FIND statement.

The current implementation of P2J fires the trigger at the latest moment possible, ie. when the content of the buffer is about to be flushed to database (RecordBuffer.flush()). In both cases, P2J will do the respective flushes as soon as the transaction blocks end their scopes, for the cases above being much earlier than P4GL.

I was not able to create a test-case with iterative statements as the buffer is automatically refreshed with the next value and the old is flushed automatically between iterations.

To match the P4GL behavior, we need to change the moment when flush() is called in P2J, the important place here being processValidate() of TransactionManager. I am not aware at this moment of the implications of removing the flush at that point and use only the indirect calls when the buffer is needed to store new info (create(), delete(), setCurrentRecord()).
Maybe the only block to trigger the flush could be the EXTERNAL_PROCEDURE ?

#2 - 01/27/2014 03:58 PM - Eric Faulhaber

- Assignee set to Ovidiu Maxiniuc

#3 - 02/03/2014 02:11 PM - Ovidiu Maxiniuc

I encountered two sentences that look interesting from the WRITE-trigger pov:

- The AVM performs validation when it leaves a field.
  What does "leaves" means ? It gets out of scope ? I noticed that when the buffer is re-filled with some new record (read from db or newly CREATEd) the old record is VALIDATEd/flushed and WRITE event triggered.
- If a field or table has been modified, the VALIDATE statement causes WRITE events and all related WRITE triggers to execute.
  I have tested a few calls and indeed, manually calling VALIDATE will fire WRITE event on 4GL if the record was altered. On the other hand, we are more aggressive in triggering WRITE than if an VALIDATE would be called before each 4GL block (CREATE without being called by some ASSIGNs won't make VALIDATE trigger WRITE event).
  We need to add support for this too, we don't flush the buffer on manual VALIDATE.

At this moment it looks like the implementation should decouple the RecordBuffer.validate() / TransactionManager.processValidate() from finally block of BlockManager.processBody() at least for RecordBuffers. Operation does not look very easy.

#4 - 02/03/2014 02:27 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> - The AVM performs validation when it leaves a field.
>   What does "leaves" means ?

This reads to me like they are referring to a UI field (like a fill-in), with which a database field has been associated. I expect this refers to a focus loss. Where is this documentation?

#5 - 02/04/2014 10:09 AM - Ovidiu Maxiniuc

Please see the last notes of VALIDATE statement, in ABL Reference rev 10.2B, page 1115.

#6 - 02/07/2014 03:04 PM - Ovidiu Maxiniuc

Write-to-db is NOT expected at the end of a block including:
- transactions,
- internal procedures
- functions

TransactionManager.processValidate():3887

- when called form BlockManager.processBody()
  - from all Commitables, RecordBuffer.validate() should NOT flush/write to database
  - the other implementations of Commitable.validate() are no-ops anyway
- when called from BlockManager.processForBody() should NOT flush/write to database.
  flush/db write will naturally occur when the current iteration changes.

RecordBuffer.endBatch():2892
vHelper.validateDirty()
- should NOT mandatory flush/write to database (at least when used to close an ASSIGN bracketing)
- however, it is used from other places from which I cannot say the same:
BufferManager.batchNotify(boolean start)
FieldAssigner.Scope.processAssignments()
OutputExtentParameter.Scope.processAssignments()
- solution: extra parameter for flush() ?

RecordBuffer.ValidationHelper.wasTouched()
- should give a rather good test for executing the flush/WRITE to database event

The write-to-db event should occur (as mentioned in previous note) when:

- buffer is needed to store new record of CREATE
- buffer is needed to store data retrieved from db
- manually with P4GL validate statement
- quit / end of program / external procedure (buffer gets out of scope)
- note:
  - if there is an assign statement that is rolled-back by an ASSIGN trigger, the record
  is still marked as dirty and it will be WRITTEN to db even it is actually unchanged

RecordBuffer.setCurrentRecord():8308 * already handles the flushing of old record when loading other from database

#7 - 02/07/2014 03:38 PM - Eric Faulhaber

OK, good work. Please go ahead and implement these changes and document the names of the test cases which you created to determine these rules (and check them into the testcases project if you haven't already).

#8 - 02/13/2014 04:20 PM - Ovidiu Maxiniuc

- File om_upd20140213a.zip added

As Eric affirmed in a mail I am *modifying some very sensitive code*.
The flush procedure (WRITE triggers and effective db persistence) is not usually block related. As mentioned in a previous note, flushing occurs automatically when the buffer is needed (new record is CREATEd or loaded from database using FINDs) or programatically, when VALIDATE statement is used for respective buffer. A particular case of flush occurs at the end of the external procedure is the buffer was altered. In fact it seems to me that this is the only case of block-related persist operation.
At all time,

- the previous copy of the flushed record is kept so that it can be available as the optional OLD buffer parameter in the WRITE trigger.
- there is some kind of touched flag. If it is not set, there is no need for a flush operation.
  When the persist operation is performed, the flag is invalidated.

All these look fine and changing flush events from end of blocks was easy to implement. The problems appear when there are UNDO statements. Which are block-related. Beside the fact that they rollback the buffer to the value from the start of the block, this is the moment when I need to set back the saved copy for old parameter of WRITE trigger and, in some cases, a silent save is needed to undo buffers that were clean. In other cases only the dirty/touched flag will suffice to force the flush at a future moment.

#9 - 03/17/2014 04:33 PM - Eric Faulhaber

Please merge your update with the latest code in bzr.

Regarding the UNDO problem, do you need to enhance or develop a new type of Reversible for this behavior?

You had mentioned separately (in email, I think) that regression testing of this update did not go well. Please elaborate here.

#10 - 03/19/2014 03:54 PM - Ovidiu Maxiniuc

- File om_upd20140319a.zip added

Code from om_upd20140213a.zip merged with bzr rev. 10493 uploaded.

The simple UNDO issue is solved in the update.
However, I discovered an UNDO, NEXT issue, not related to current task; I also think cannot be fixed by implementing a new Reversible. The following behavior occurs with existing p2 code from bzr:

```
for each book:
    display isbn.
    isbn = "abc".
    undo, next.
end.
```

Normally, this should iterate all records and display the isbn s. It does that on P4GL.
However, on P2J, this will loop continuously on first item from book, because at undoNext(), the current transaction is reset, causing the record pointer of ProgressiveResults from PreselectQuery to be reset, too. Here is the important part of stack trace:

```
  at com.goldencode.p2j.persist.ProgressiveResults.cleanup(ProgressiveResults.java:565)
  at com.goldencode.p2j.persist.ResultsAdapter.cleanup(ResultsAdapter.java:296)
  at com.goldencode.p2j.persist.PreselectQuery.resetResults(PreselectQuery.java:2698)
  at com.goldencode.p2j.persist.AdaptiveQuery.resetResults(AdaptiveQuery.java:2274)
  at com.goldencode.p2j.persist.AdaptiveQuery.sessionEvent(AdaptiveQuery.java:1408)
  at com.goldencode.p2j.persist.Persistence$Context.notifySessionEvent(Persistence.java:5321)
  at com.goldencode.p2j.persist.Persistence$Context.closeSessionImpl(Persistence.java:5260)
  at com.goldencode.p2j.persist.Persistence$Context.closeSession(Persistence.java:5144)
  at com.goldencode.p2j.persist.Persistence$Context.endTransaction(Persistence.java:4776)
  at com.goldencode.p2j.persist.Persistence$Context.rollback(Persistence.java:4662)
  at com.goldencode.p2j.persist.Persistence.rollback(Persistence.java:3550)
  at com.goldencode.p2j.persist.BufferManager$DBTxWrapper.rollback(BufferManager.java:2192)
  at com.goldencode.p2j.util.TransactionManager$WorkArea.notifyMasterCommit(TransactionManager.java:5520)
  at com.goldencode.p2j.util.TransactionManager$WorkArea.access$47(TransactionManager.java:5497)
  at com.goldencode.p2j.util.TransactionManager.processRollback(TransactionManager.java:4718)
  at com.goldencode.p2j.util.TransactionManager.rollbackWorker(TransactionManager.java:1702)
  at com.goldencode.p2j.util.TransactionManager.rollback(TransactionManager.java:1589)
  at com.goldencode.p2j.util.BlockManager.undoNext(BlockManager.java:5914)
  at com.goldencode.p2j.util.BlockManager.undoNext(BlockManager.java:5892)
  at com.goldencode.testcases.write_delay.TestRoot$1$1.body(TestRoot.java:46)
  at com.goldencode.p2j.util.BlockManager.processForBody(BlockManager.java:7095)
  at com.goldencode.p2j.util.BlockManager.forEachWorker(BlockManager.java:8640)
  at com.goldencode.p2j.util.BlockManager.forEach(BlockManager.java:3717)
  at com.goldencode.testcases.write_delay.TestRoot$1.body(TestRoot.java:29)
```

Should we endTransaction() in this event ? Perhaps this should be handled in a new task, anyway.

Regarding the regression test for om_upd20140213a.zip.
The gso_ctrlc_tests were OK and the gso_ctrlc_3way_tests had some irrelevant failings.
On the other hand, gso_tests had 100+ failures, gso_rfq_tests 4 and tc_tests 50+.
Some examples of found issues:
- gso_29/step 53: Records selected: 502 instead of 475
- gso_108/step 72: Records selected: none instead of 1
- gso_114/step 44: Client reset to login screen (lost track inside the job/job63-r.p, job63-r.p/j63hmain.p)
However, I have some exceptions logged in the server log:

```
at com.goldencode.p2j.persist.Persistence.commit(Persistence.java:3531)
```

```
at com.goldencode.p2j.persist.RecordBuffer.flush(RecordBuffer.java:7701)
at com.goldencode.p2j.persist.RecordBuffer.setCurrentRecord(RecordBuffer.java:8420)
```

causes some org.hibernate.TransactionException: nested transactions not supported.
I am continuing investigations to see what I'm doing wring and if I can reproduce this locally in a test case, hopefully, this will eliminate much of other failed tests.

#11 - 03/27/2014 04:22 PM - Ovidiu Maxiniuc

It seems that I have implemented wrong the new write to database event.
My initial approach and the fix for it was at buffer-level.
However, after some more researches I realized that the records should be monitored against the block events.
Practically, the some parts of implementation should be moved (I am working on this) from RecordBuffer to a better place - BufferManager that also tracks the count of each DMO usage.
Mainly these are the events that must be handled for a record r:

- blockEnter() - stack the copy of r's latest persisted DMO (will be used at rollback)
- decrementDMOCount() - if 0 is reached the record MUST be saved to persistence. The last buffer will be used in WRITE event.
- validate() - if record is changed in memory (even the change is done using other buffer) save it. Use the current buffer as WRITE event parameter.
- blockLeave() - in UNDO mode, the latest persisted DMO will rollback to stacked copy if different than the latest value (this happens independently from the actual content of the buffer). The copy will be re-set to persistence.
- each time an assign happens on any of the buffers referring it, the record is marked as dirty in this block. This flag also must be restored at blockLeave().

#12 - 04/29/2014 04:33 PM - Ovidiu Maxiniuc

- File om_upd20140429a.zip added

- File server.log.zip added

Attached the update form this morning that generated the also attached server log.

The sessions are still 'synchronized' with blocks and all Commitables are committed or roll-backed when a block ends
I don't think this s correct. The undo / rollback operations are normal at this point. Apparently the commit part must not execute or should let the session alive.

On the other hand, IIRC, Hibernate Sessions are intended as a short-lived objects: use-and-release, Hibernate is responsible for their long-term management.

I'm preparing a new update with a dmo-counter fix that was damaged in this update and put it to a preliminary ctrl+c test.

#13 - 05/02/2014 03:48 PM - Ovidiu Maxiniuc

- File om_upd20140502b.zip added

- File 2222-testcases.zip added

I attached the last version of implementation. The files contain extra logging code.
Also the current set of testcases. I did not re-check all of them, but the latest I used were successful. At the end of the main files, they contain the expected output from P4GL. Some of them also contain a header with intended cases to test.

#14 - 05/05/2014 10:15 AM - Ovidiu Maxiniuc

Most likely, you already know much of following, but I believe is better to repeat them, in the context:

To explain in short what I discover how Progress handles flushes of modified/transient records is that it is delaying the the write of a record to database as long as possible. This means, as long as at least a buffer holds record X, the record will not be flushed. We already have the code that counts the buffer referring a DMO, but it is used only for freeing the (memory) resources occupied by such record.

Of course, it is possible that some buffers are still 'occupied' at the end of a procedure. When the procedure/program block ends, the buffer content is released and if it was the last buffer holding the record a WRITE operation will occur later, while releasing buffers, AFTER the program's code is finished. If multiple buffers are released at this moment, it looks like the default buffer is released last, so that the buffer in WRITE trigger is the default one (at least in my tests - I don't have a proof for this).

The same happens on other cases when the DMO of a buffer is changed: when LOADing a new record (via FIND, even if it fails because the sought record is not found; and CREATE, which will populate the buffer with a fresh transient DMO).

There are two cases when the P4GL programmer can manually flush the records to database:

- VALIDATE on any of the buffers that holds a record. If the record is OK, the WRITE operation takes place (including WRITE triggers) and all buffers.
- if some buffer is the only one referring a DMO, calling a RELEASE on it will also trigger WRITE. However, this is rather a consequence of the previous points (because RELEASE will decrement the buffer refcount to DMO, and since it reaches 0, it will be VALIDATEd and flushed).

Note: triggers are not fired by the temp-tables. In fact it is a compile-error to set a trigger on such a table. The code from bzr did not explicitly forbid that, based on the assumption that the processed code is P4GL valid, and since so such trigger was defined, it will not be found in the DatabaseTriggerManager data structures. As an optimization, I added a test for this in upd20140502b.

#15 - 05/06/2014 03:43 AM - Ovidiu Maxiniuc

I realized I forgot to mention one thing that I put aside because it did not fit well with my researches.

My first approach was to use the ChangeBroker. The notifications about changes in DMO and block levels made this the best candidate for handling them for delaying the write trigger. The change event was the most important. The problem arose when the event was dispatched to all listening record buffers, that turned me back to initial problem (from the point of view of a document-view architecture, handling the event at document level, not in each view).

I was interested in saving only some particular snapshots (at the moment of WRITE event) - that is rather asynchronous compared to (rather many) snapshots generated by field changes. Also, on an undo event, the roll-backing should restore of the last persisted snapshot of DMO, which is usually different from the snapshot at the beginning of the block.

Also there was some additional information to be saved along this snapshot, the flag that marks if the DMO was altered since last WRITE.

1st scenario

```
FIND r.
ALTER r.
VALIDATE r.        /* causes WRITE */
   {
      ALTER r.
      VALIDATE r. /* causes WRITE */
      UNDO.
   }
VALIDATE r.        /* WRITE not needed */
```

2nd scenario

```
FIND r.
ALTER r.
VALIDATE r.        /* causes WRITE */
ALTER r.
   {
      ALTER r.
      VALIDATE r. /* causes WRITE */
      /* the old-buffer value is the snapshot in 1st WRITE */
      UNDO.
   }
/* at the end of this procedure a WRITE will trigger automatically because there are still pending changes unf
lushed (2nd Alter, before entering the transaction block);
the old-buffer value is the snapshot in 1st WRITE, again */
```

In current implementation, the 'flush' of ChangeBroker means the flushing of a Hibernate Session on which the DMOs have already committed within the current Persistence.

#16 - 05/13/2014 04:53 PM - Eric Faulhaber

- Assignee changed from Ovidiu Maxiniuc to Eric Faulhaber

#17 - 08/13/2014 05:47 PM - Eric Faulhaber

- File ecf_upd20140813a.zip added

The attached update fixes a defect which enabled write triggers to corrupt the BlockManager's context-local state.

Greg, please review.

#18 - 08/14/2014 08:41 AM - Greg Shah

Code Review 0813a

I am fine with the changes.

#19 - 09/05/2014 12:06 AM - Eric Faulhaber

- File ecf_upd20140903a.zip added

Ovidiu, would you please review the attached update? It goes way beyond the write trigger fix to deal with the incorrect timing of our implicit validation as well.

#2371 describes the trigger behavior I am trying to match.

In terms of validation, the idea is to validate:

- upon changes to an index (this means upon a change to any field in an index for an existing record, but upon a change to all fields in an index for a new record);
- upon the release of a record from a buffer;
- upon explicit validation (VALIDATE statement);
- upon the earlier of the end of a full transaction or the end of a buffer's scope.

The last one is new, and dragged in a lot of other changes.

In my testing of the 4GL behavior, I found that our dirty database implementation was wrong in several ways. That implementation is designed to leak changes to indexes across transactions, to emulate Progress' leaky transaction isolation. I found that I needed to share awareness of changes in certain cases across buffers in the same session as well. This led me to add a new record to the dirty database immediately upon its creation (we used to wait until it was fully validated and flushed to the database).

Previously, we used the BufferFlushQueue to flush all buffers backed by the same table before any kind of query, so that the query could find a record newly created by another buffer. This was too aggressive in some cases and had to be removed, as it was triggering validation errors at the wrong times. The creating buffer is now flushed according to the validation rules listed above.

There are some known limitations, none of which were severe enough to hold up this update:

- I found cases where a query which does not use an index to sort its results can trigger the validation/flush of a new record in a separate buffer, earlier than it normally would happen (presumably to load the fields on which it needs to sort). I plan to address this in a future update.
- In certain cases (like a for each loop with a full transaction scope), a find trigger will be fired inside a transaction, whereas Progress reports the same trigger to be outside the transaction. This happens because of our implementation of these queries, which places the next method within the body of the block, instead of outside it.
- Errors raised in find triggers are not handled correctly yet, because we are launching them like UI triggers, but the launching code needs to be specific for database triggers. Actually, this probably applies to all database triggers, but I found it with a find trigger.
- Deleting a buffer object by its handle causes the buffer to flush prematurely in our implementation. This needs to be deferred.
- The removal of the BufferFlushQueue and the fact the all dirty database operations are no-ops when using temp-tables means we have no means at the moment to share awareness of new records in temp-table buffers with other buffers in the same session. Fortunately, it is much less likely to find code which relies on this, because the technique of creating a record in one buffer and then loading it in another before it is fully validated is more prevalent with persistent tables, where the developer was trying to avoid concurrency issues. Nevertheless, I intend to address this in a future update.

Various incarnations of the update have gone through many rounds (over 25) of regression testing. This version of the update currently regresses only one test scenario consistently (down from hundreds a few weeks ago). It is a particularly painful one to debug, as it takes several hours to set up, and apparently will only occur when there are some (as yet unknown) conditions in a separate session -- I spent all day debugging this one today, and in the end it worked normally, so there's some concurrency dependency there.

Anyway, I am continuing to look for the cause, but hopefully it is unlikely to result in a substantial change in the end, so I'd like you to begin your

review now, even though it's not quite finished. The review may take a while -- the changes are in a particularly complicated area of the persistence framework.

Let me know if you have any questions or concerns.

#20 - 09/05/2014 06:19 PM - Ovidiu Maxiniuc

I have not finished yet reviewing your update 20140903a, only a side-by-side compare with current bzr. At first impression is looks good. Some impressions and questions:

- I like the new TriggerTracker class that elegantly centralize information about when the trigger availability that was scattered in RecordBuffer. IIRC, I needed a stack of oldDMO to keep those 'waypoints' in UNDO events. Is it implemented some other way, or is it not really necessary ?
- I observed the 'prototype' solution of deferring the change events, validation and flushing to be processed to the initial RecordBuffer that created the record. I went for a document-view implementation but this is a simpler faster solution.
- the dirty package I must redo, I did not fully comprehend all changes.
- one question: what do you mean by *"check if write trigger fires for an extent field"* (the TODO at RecordBuffer.java:10241)?
  - WRITE triggers are fired at record level
  - the ASSIGN triggers (for a field) are not allowed to be defined for EXTENT fields.
- one more optimization that can be added. Knowing that triggers do not make sense for temp-tables, a quick-out test for permanent tables (in isTriggerEnabled maybe?) could save a few CPU ticks (by avoiding unneeded map lookups and other processing).
- I saw that the update contains other changes (TemplateWorker & TempTableBuilder)

I will be back with a detailed review. I intend to run the update with my old testcases (although probably you already did that).

#21 - 09/05/2014 09:59 PM - Eric Faulhaber

- File ecf_upd20140905a.zip added

Ovidiu Maxiniuc wrote:

> I have not finished yet reviewing your update 20140903a, only a side-by-side compare with current bzr. At first impression is looks good. Some impressions and questions:
>
> - I like the new TriggerTracker class that elegantly centralize information about when the trigger availability that was scattered in RecordBuffer. IIRC, I needed a stack of oldDMO to keep those 'waypoints' in UNDO events. Is it implemented some other way, or is it not really necessary ?

I don't think it is necessary, because we are copying the state of the current record at the moment before the first change is made to it. Subsequent sub-transaction commits and rollbacks may affect the current record, but shouldn't affect the old state. If the scope in which the copy was made is itself rolled back, the copy is dropped and the write trigger won't fire, because at that point any changes made at the current or more deeply nested scopes will have been undone.

> - I observed the 'prototype' solution of deferring the change events, validation and flushing to be processed to the initial RecordBuffer that created the record. I went for a document-view implementation but this is a simpler faster solution.

I'm not sure what you mean. Are you talking about the code that uses creatingBuffer, or something else?

> - the dirty package I must redo, I did not fully comprehend all changes.

Sorry, this is one of the most complicated areas of the persistence runtime, because it has to emulate a very bizarre quirk of Progress without the same backing infrastructure that Progress uses. The Progress database leaks uncommitted changes to indexes across transactions, because it is essentially an index engine with no notion of MVCC. When a record is inserted, removed, or moved within an index in one transaction, other transactions can see this change immediately, though they can't necessarily access other, non-indexed information until the first transaction is committed. These uncommitted changes can affect the content and sorting of search results and checking against unique constraints in unrelated transactions.

In our implementation using an MVCC database like PostgreSQL, where the indexes are encapsulated within the database, we can't walk them directly. So, we have to emulate this ridiculous behavior using an embedded H2 database which tracks these "dirty" changes as they happen. To do this, we have to track index changes faithfully, validate them at exactly the right moments, and synchronize updates to the "virtual" state of these indexes in the dirty database very carefully. Changes to the timing of validation and flushing had ripple effects throughout this implementation. Hence, the complicated set of changes.

- one question: what do you mean by *"check if write trigger fires for an extent field"* (the TODO at RecordBuffer.java:10241)?
  - WRITE triggers are fired at record level
  - the ASSIGN triggers (for a field) are not allowed to be defined for EXTENT fields.

I was referring to whether a change to an extent field would necessitate the later firing of a write trigger, the way a change to a scalar field does. The question occurred to me because I saw you had determined assign triggers were not allowed for extent fields. Do you have a test case that with a write trigger which changes only an extent field? I forgot to follow up on this TODO and write one.

- one more optimization that can be added. Knowing that triggers do not make sense for temp-tables, a quick-out test for permanent tables (in isTriggerEnabled maybe?) could save a few CPU ticks (by avoiding unneeded map lookups and other processing).

Good idea. I didn't realize you could not set a trigger on a temp-table. Does this hold true for session triggers, or just schema triggers?

- I saw that the update contains other changes (TemplateWorker & TempTableBuilder)

Oops, TempTableBuilder should not have been included. I was testing an interim update from Vadim and zipped it in by mistake. XmlFilePlugin and TemplateWorker are in there because I told Constantin I would implement a separate suggestion of his and roll it into my next update, but they're unrelated.

I will be back with a detailed review. I intend to run the update with my old testcases (although probably you already did that).

Actually, I did not. I created a new test case framework as this task became entangled with the validation and then dirty database implementations. I would appreciate it if you would run your old test cases against it. I'm attaching a new version which fixes defects in TriggerTracker and DatabaseEventType, removes TempTableBuilder, and has temporary debug code added to RecordBuffer.

Thank you for reviewing this.

#22 - 09/08/2014 05:07 PM - Ovidiu Maxiniuc

- File testcases 2014.09.08.zip added

Eric,

I reviewed closely your last update. Changes make sense. I did not find any issues except for a few javadoc (DirtyShareContext.getDirtyInfo() and the two implementations lack a blank line for separating params and @return) and in @DirtyShareMultiplexerImpl.java:201 javadoc comment is open twice.

The answers to your questions:

- indeed, creatingBuffer I was referring in 2nd item from note 20. However, this must not be used only for created records. I mean, the same object should handle triggers on a record, no matter how they get in memory including the loaded ones. Please see two-buffers-for-same-record.i the attached archive. P4GL will fire the WRITE trigger only once (when the record loaded is flushed, using the first buffer that loaded the record as far as I could infer). We are triggering it twice, probably the record is written twice to database, once for each buffer referring the record (book and x-book).
- ASSIGN triggers cannot be defined on extent fields this is a compile-time error (Fields with extents may not have ASSIGN triggers. (3433)). I did not implemented this because I was told that the input code for conversion is P4GL correct (or at least it passed the P4GL compiler).
- P4GL fires the WRITE triggers on record if any field has changed, including the EXTENT ones. P2J does not fire WRITE triggers in if only EXTENT fields are changed. (See the trigger-write-from-extent.i in the archive.)
- Trying to add a trigger to a temp-table is a compile error (You may not associate database triggers with workfiles or temp-tables. (3335)). Error message not added for same reason as for ASSIGN to EXTENT fields. Schema triggers cannot be added as the temp-tables are not available at that time. I don't think that there's a way to add a database trigger to a dynamic temp table.
- Here are other testcases that failed with this update:

1. two-buffers-for-same-record2.i. Again, this proves that flushes/WRITE are triggered once per record. Validating a buffer will validate all that refer the same record. One more UI issue here: in P4GL, you can see all 29 lines. In P2J only the last 3 (24-26) are visible, that's why I used the stream for recording the output.
2. write-test5.i I believe the validation happens too early. In P4GL the entire procedure is executed, P2J stops with ** book already exists with book-id 0. (132) error.
3. write-trigger-test5.i This is the 'original' testcase described in note 1. The first WRITE is executed at correct moment (step 9). The second is still broken (step 14 instead of 16).
4. complex-with-trigger-logs.i In cleanup part the @WRITE@s are executed correctly. However, the one from block 1.1 occurs too late, in block 1.2, the last one (should have been in line 29) is never executed.

The attached archive contains the data definitions and data for table Book. I believe it is slightly different than the one from repository. The files need to be processed one at a time. I could not design a better framework. I give up of schema triggers as they were not important from this task point of view. The test-cases have the expected output (rendered by P4GL) at the end of file, in a comment. The numbers represent the buffer ids and should normally be ignored. If you need I can render the outputs in some side by side. Please let me know if I can help otherwise.

#23 - 09/12/2014 02:04 PM - Eric Faulhaber

- File ecf_upd20140911a.zip added

Attached update has passed regression testing and is checked into bzr revs. 10609 (all changed/new files) and 10610 (removed file I forgot to commit w/ 10609).

Ovidiu: it does not address all of the concerns from your review, but it should have fixed the multiple write trigger firing problem. I will continue working on the other issues you reported, thanks.

#24 - 09/13/2014 02:52 PM - Eric Faulhaber

- Status changed from New to WIP

- % Done changed from 0 to 80

#25 - 03/30/2016 12:54 PM - Eric Faulhaber

- Status changed from WIP to Hold

- Target version deleted (Milestone 11)

The scope of this issue grew way beyond the originally reported problems that impacted M11. The original problems have long since been resolved, so I am moving this issue out of M11.

#26 - 03/31/2016 04:27 AM - Paul E

> The original problems have long since been resolved

Does this mean that the behavioural differences as noted by Ovidiu in his table in note 1 have been resolved?

#27 - 05/03/2016 05:17 AM - Paul E

I'd appreciate an update on this if that's ok?

See note 26.

#28 - 05/03/2016 10:15 AM - Eric Faulhaber

Paul Eames wrote:

> > The original problems have long since been resolved
>
> Does this mean that the behavioural differences as noted by Ovidiu in his table in note 1 have been resolved?

Yes, unless we've regressed something, I believe that's the case.

Ovidiu, would you please confirm?

#29 - 05/04/2016 08:02 AM - Ovidiu Maxiniuc

Yes, the cause for this issue is fixed now. The WRITE triggers are fired at the right moment up to step 13 of the test procedure.

However, there seems to be another issue now. The last WRITE (step 14) is triggered to late (after DELETE).
I did a short debugging session and the cause seems to be an optimization. At the moment the FIND for isbn = 222 is executed, the record with matching isbn is already in buffer. 4GL will fire the WRITE trigger in this case, but P2J is just happy to keep the sameRecord in buffer (RecordBuffer:9534), fire FIND trigger and continues to next statement. The actual WRITE will be performed by P2J only after the DELETE trigger after the record is removed from database and discarded from P2J persistence (this is a little odd, anyway).
I guess we should open a separate task for this new behavior.

LE:
For this particular case adding a maybeFireWriteTrigger(); before exiting the RecordBuffer.setCurrentRecord() at line 9534 will cause P2J to generate the same output for the testcase as 4GL.

#30 - 05/04/2016 08:29 AM - Paul E

Thanks for the update Ovidiu.

The thing here that interests me most is that you have these tests that you run against your book database. Is this a common database that you use at Golden Code for regression testing? Do you have lots of useful tests written against this database?

I wonder if it would help for us to implement a CI system that we can use to exercise these tests. Might be worth discussing.

#31 - 05/04/2016 09:04 AM - Ovidiu Maxiniuc

Paul Eames wrote:

> Thanks for the update Ovidiu.
>
> The thing here that interests me most is that you have these tests that you run against your book database. Is this a common database that you use at Golden Code for regression testing? Do you have lots of useful tests written against this database?
>
> I wonder if it would help for us to implement a CI system that we can use to exercise these tests. Might be worth discussing.

Paul,
We have a test database we are (at least me) testing features we work on. The database is very small, only a few tables, a base for constructing testcases and not starting each time from the scratch. The usage is very non-standardized and depends on the task at hand. Usually I add new short-lived tables or fields to existing tables a needed by the respective feature. It is not synchronized.
The testcase sources are stored in a very non-standardized repository. They are initial testcases for a new feature or isolations of faulty ones. Usually we put the path in the current task. A CI system is generally a good idea, but I think it would require too much effort here.
I think Greg is the best person to discuss on this subject.

#32 - 04/08/2019 05:10 PM - Eric Faulhaber

- Related to Bug #4035: newly created record is not flushed at the right time added

Files

write-trigger-test5.p	2.36 KB	01/08/2014	Ovidiu Maxiniuc
write-trigger-test5.p.lst	3.76 KB	01/08/2014	Ovidiu Maxiniuc
om_upd20140213a.zip	216 KB	02/13/2014	Ovidiu Maxiniuc
om_upd20140319a.zip	222 KB	03/19/2014	Ovidiu Maxiniuc
om_upd20140429a.zip	186 KB	04/29/2014	Ovidiu Maxiniuc
server.log.zip	17.7 KB	04/29/2014	Ovidiu Maxiniuc
om_upd20140502b.zip	277 KB	05/02/2014	Ovidiu Maxiniuc
2222-testcases.zip	24.2 KB	05/02/2014	Ovidiu Maxiniuc
ecf_upd20140813a.zip	71.3 KB	08/13/2014	Eric Faulhaber
ecf_upd20140903a.zip	350 KB	09/05/2014	Eric Faulhaber
ecf_upd20140905a.zip	335 KB	09/06/2014	Eric Faulhaber
testcases 2014.09.08.zip	7.04 KB	09/08/2014	Ovidiu Maxiniuc
ecf_upd20140911a.zip	336 KB	09/12/2014	Eric Faulhaber