Base Language - Bug #5203

Temp Table - serialization/transaction scope/pk

03/16/2021 05:52 AM - Marian Edu

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

Description

History

#1 - 03/16/2021 05:52 AM - Marian Edu

There seems to be a difference in how 4GL is handling serialization/transaction scoping for temp-tables. First I though it has to do with the presence of a primary key but it's just the same with a table that has no unique index defined.

In fwd the 'uncommitted' records are simply not visible for serialization - TemporaryBuffer.readAllRows do not 'see' those. Calling validate doesn't help either although imho it should force the record to be 'flush', this seems to happen however for a temp-table that has a unique index defined and a value is set for the field that is part of the index (validate should more or less do the same thing, validate all mandatory fields, unique constraints).

The only things that works is to force a transaction block, then the updated records became visible for serialization.

Quick test in oo/json/quirks/table_serialize.p.

#2 - 03/16/2021 07:38 AM - Constantin Asofiei

Eric/Ovidiu: the above looks like a temp-table flush issue.

#3 - 03/16/2021 08:43 AM - Greg Shah

- Assignee set to Eric Faulhaber

#4 - 03/16/2021 12:22 PM - Ovidiu Maxiniuc

I do not think the problem here is the transaction. It seems to me that this is only related to serialization methods. The temp-tables are auto-committed if they are not indexed with unique indexes, that's why those tables seem to be always flushed and the others require the full transaction.

The 'uncommitted' records are just one for each each buffer. For the record to get to database the record must be flushed and I am not sure if the validate should do this.

The current implementation of serialization, a faster, low-level approach is used. The buffer is not affected, it will keep the current content. This might be the actual issue. Issuing a flush() or release() if the buffer content must not be preserved will fix the problem. However, in case the invocation is done on a temp-table object, I assume all its buffers must do this operation. Or only the default one? These questions need answers which can be obtained only from tests.

#5 - 03/16/2021 12:53 PM - Marian Edu

Ovidiu Maxiniuc wrote:

> I do not think the problem here is the transaction. It seems to me that this is only related to serialization methods. The temp-tables are auto-committed if they are not indexed with unique indexes, that's why those tables seem to be always flushed and the others require the full transaction.

This is not the case, for a temp-table that doesn't have any unique index thee 'current' record is not visible to serialization so it's really not alway flushed.

> The 'uncommitted' records are just one for each each buffer. For the record to get to database the record must be flushed and I am not sure if the validate should do this.

I'm pretty sure it should (https://docs.progress.com/bundle/openedge-abl-reference-117/page/BUFFER-VALIDATE-method.html) - the write event really means the record has written, the write triggers fires, if any unique constrains are violated by current record values this throws an error, however the record remain in buffer and it's lock status is not released (it is not a RELEASE).

> The current implementation of serialization, a faster, low-level approach is used.

Don't know what that means exactly but as said TemporaryBuffer.readAllRows doesn't see that uncommitted record, and that should be the same default temp-table buffer.

> The buffer is not affected, it will keep the current content. This might be the actual issue. Issuing a flush() or release() if the buffer content must not be preserved will fix the problem. However, in case the invocation is done on a temp-table object, I assume all its buffers must do this operation. Or only the default one? These questions need answers which can be obtained only from tests.

I don't think the buffer should be released when any serialization routine is being executed, and beside all those temp-tables are NO-UNDO option so why would forcing a transaction block make any difference? I will try to extend the tests tomorrow.

#6 - 03/16/2021 01:42 PM - Eric Faulhaber

Marian Edu wrote:

> Ovidiu Maxiniuc wrote:
>
>> The 'uncommitted' records are just one for each each buffer. For the record to get to database the record must be flushed and I am not sure if the validate should do this.
>
> I'm pretty sure it should (https://docs.progress.com/bundle/openedge-abl-reference-117/page/BUFFER-VALIDATE-method.html) - the write event really means the record has written, the write triggers fires, if any unique constrains are violated by current record values this throws an error, however the record remain in buffer and it's lock status is not released (it is not a RELEASE).

We may need to refine our recent fix for #5056, which danced around this area. In that case, the fix was to not update the internal DMO state for an explicit validation (as with BUFFER-VALIDATE). But the tests for that were designed to fail validation, so we weren't really focused on the flushing and triggers part after a successful validation. We'll have to review what happens in terms of the flush in cases where explicit validation succeeds. IIRC, at the moment, we just report the result of an explicit validation, but we do not flush (or fire WRITE triggers).

Do we already have a standalone test case (or small set of cases) which hits this point? I mean, without pulling in the entire test suite?

#7 - 03/16/2021 03:42 PM - Ovidiu Maxiniuc

This is a bit strange: suppose we have a record in buffer but, since the record was not validated, it may be invalid. If this is the case, what happens? What is saved?

Since we are talking about temp-tables, the triggers are not an issue. The triggers are defined only for database/persistent tables.

#8 - 03/16/2021 04:43 PM - Ovidiu Maxiniuc

I created the following test:

```
DEFINE TEMP-TABLE ttestt
   FIELD f1 AS INTEGER
   FIELD f2 AS CHARACTER
   INDEX uidx1 AS PRIMARY UNIQUE f1.

CREATE ttestt. f2 = "0". // f1 = 0
CREATE ttestt. f2 = "1".    f1 = 1.
CREATE ttestt. f2 = "2". // f1 = 0, again, but not validated/flushed yet

MESSAGE "Before:" RECID(ttestt) f1 f2.
MESSAGE BUFFER ttestt:WRITE-XML("file", "how-many-0s.xml").
MESSAGE "After:" RECID(ttestt) f1 f2.

FOR EACH ttestt:
    MESSAGE RECID(ttestt) f1 f2.
END.
```

Firstly, notice that there are two records with f1 = 0. One already flushed to storage, one in buffer at the moment the WRITE-XML is invoked.

When it is executed, error 132 is shown (ttestt already exists with 0.), then 13093 (Write temp-table data failed for WRITE-XML). So, yes, the current buffer was (at least) validated. FWD does not do this.
Then error 91 is shown (No ttestt record available), twice. This is generated by the After: message, because the buffer is empty.
The final loop only shows two records, as expected.

To test whether the record was empty because of the error, I removed the creation of the 3rd record. In this case, although no errors are generated during xml serialization, the after message still give the same two 91 error. So, the buffer is released. This is not expressly said in documentation, just that the content is unreliable.

So, to fix the issue, a simple release called on the working buffer before invoking TemporaryBuffer.readAllRows should be sufficient. Note that the output xml file is created even if the method fails and the content just unfinished.

Then I added a secondary buffer for same table:

```
DEFINE BUFFER ttestt-2 FOR ttestt.
```

From my tests, all active buffers of the table are validated/flushed at the moment the write-xml method is called, but only the one on which the method was invoked is release-d. If the method of the temp-table objects is invoked, the default buffer is release-d. If the method is invoked on a dataset handle, the buffer which was added to dataset is release-d.

#9 - 03/17/2021 06:55 AM - Marian Edu

I've updated the tests in oo/json/quirks/table_serialize.p and the behavior is rather confusing :(

For a temp-table with PK when using additional buffer defined for the table and create a record in temp-table (default buffer) and one in the extra buffer that violates the constraint.

Write-xml for the temp-table raise no error, only content of default buffer is written, default buffer is released, extra buffer remain available.

Write-xml for the extra buffer raise error, extra buffer remain available.

So, it looks like it's doing a validate on the buffer but only on that particular buffer - if that does not throw it's using a query on the buffer to go through all records so that is leaving the buffer unavailable at the end, if validation errors the buffer remains available and not changed.

Now, if the buffer is part of a dataset this changes as it will always throw error if there are dupe records in other buffers for the same temp-table even if only one buffer is part of the dataset. It looks like it's doing validate for all buffers defined for the temp-table(s) that are part of the dataset. The validation order seems more or less random although I might be missing something cause the tests are idempotents so whatever is doing is always doing it in exact same order :(

What it does if there are validation errors is it deleted the record from the buffer where the validation failed and stop processing (it does not validate/purge all remaining buffers).

I've tried to add more tests and from what I can see the validation always starts with additional user defined buffers first, default buffer is always left alone so it's probably the last one checked which is odd because the order seems to be the one in which the 'uncommitted' records were created but excluding the default buffer. Aka, if we create in default buffer, then buffer 1 then buffer 2 it will fail validation on buffer 1 so the uncommitted record in that buffer will be deleted. If we create in default buffer, then buffer 2 and last in buffer 1 it will now fail in buffer 2 validation so that is the one that is being purged... why not the default buffer since the first record is created there beats me :(

This is for `write-json`, most probably same is done for xml serialization but read method in JsonObject/JsonArray is a bit different :(

#10 - 03/17/2021 12:38 PM - Ovidiu Maxiniuc

Marian Edu wrote:

> I've updated the tests in oo/json/quirks/table_serialize.p and the behavior is rather confusing :(

Welcome to the ~~clob~~ (pardon my joke) club. Indeed, the black-box method does not always give straight answers.

> For a temp-table with PK when using additional buffer defined for the table and create a record in temp-table (default buffer) and one in the extra buffer that violates the constraint.
> Write-xml for the temp-table raise no error, only content of default buffer is written, default buffer is released, extra buffer remain available.

Well, this does raise the error for me. I have just re-tested it now. I am running OE 11.6.3.

> Write-xml for the extra buffer raise error, extra buffer remain available.

Well, this is true, in one condition. If there are no unique keys already saved. From what I see, when the write-xml is invoked, all buffers are validated. The peculiarity is that the default buffer is the last one. Since the extra buffer was validated/flushed first, it encountered no unique-exception. The default buffer is last so when the validation is performed, OE will detect the collision with the record just saved.
OTOH, if the condition is not met (that is, there is already a conflicting record stored in temp-table), then OE will complain about the secondary buffer (error 132), then inform that the write-xml method failed (error 13093). At this point the operation ends, but the default buffer still contains a conflicting record. The procedure continues and the next time the default buffer is released (either because a FOR is about to be executed, like in my example above, or because the procedure scope ends) its content is validated and error 132 is shown again 3 times (I do not know why 3 time but this I noticed in #5056) for it (default buffer).

> Now, if the buffer is part of a dataset this changes as it will always throw error if there are dupe records in other buffers for the same temp-table even if only one buffer is part of the dataset. It looks like it's doing validate for all buffers defined for the temp-table(s) that are part of the dataset. The validation order seems more or less random although I might be missing something cause the tests are idempotents so whatever is doing is always doing it in exact same order :(

I did not run extensive tests with datasets, but I expect it follows the same rules.

> What it does if there are validation errors is it deleted the record from the buffer where the validation failed and stop processing (it does not validate/purge all remaining buffers).

No. It seems to me that the buffers are validated in some order, the default buffer being the last one. If a conflict is encountered, the process stops at that buffer. The buffer which passed validation were already flushed to storage, the 'faulty' buffer get discarded/purged (I guess this better describes it, not the same as 'released' as I said in my previous note) and the rest of buffers in the list remain untouched, possibly with unique index collisions. But this is not an issue as the write-xml method end at this moment and they will have another chances to be validated/flushed. It's kind of logical now.

> I've tried to add more tests and from what I can see the validation always starts with additional user defined buffers first, default buffer is always left alone so it's probably the last one checked which is odd because the order seems to be the one in which the 'uncommitted' records were created but excluding the default buffer. Aka, if we create in default buffer, then buffer 1 then buffer 2 it will fail validation on buffer 1 so the uncommitted record in that buffer will be deleted. If we create in default buffer, then buffer 2 and last in buffer 1 it will now fail in buffer 2 validation so that is the one that is being purged... why not the default buffer since the first record is created there beats me :(

Maybe the results of my investigation above shed some light here?

One more conclusion:
It seems that the buffers for a table are stored in LIFO structure. This is logically the dynamic list which provides the fastest 'add' operation. Naturally, the default buffer is always created and probably gets pushed into the stack first. The other buffer will be pushed in the order they are declared. When the buffers have to be processed, the buffers are iterated in the natural order given by this list which will always end with the default buffer which was pushed first.

#11 - 03/18/2021 09:15 AM - Marian Edu

Ovidiu, for temp-table I can add the 'release' for default buffer buffer in JsonExport but since this behavior is probably same in xml serialization it might be a better idea to simply flush the buffer in readAllRows instead?

As for flushing all buffers of a temp-table (TemporaryBuffer is what we have available), the logic there is not really visible for the naked eye, too many interfaces and getting hold of the one implementation (@BufferImpl) is not always obvious.

The whole 'persistence' stuff is quite sensitive so it's probably we stay away from it all together, did a quick test to see how query/for each behaves and there are differences between 4GL and FWD - oo/json/quirks/table_scope.p.

While in 4GL records in different temp-table's buffers can be created and have data that violates the unique index constraint doing a for each on each of those buffers will indeed flush the record and in the end the buffer is not available anymore. Now, if there is no unique index defined 4GL seems to flush the records automatically so when one open the query (for each) all records are visible. If the table has unique indexes then only the record in the respective buffer is flushed (successfully) and all others are not visible to the query. Most probably it tries to flush all buffers of the temp-table but ignore the errors if any, that's why for a table with no unique index all records are visible.

However, if flushing the buffer used in the query/for each raise error that is thrown back and the buffer remains with the invalid/uncommitted record.

No idea how for each works in FWD but for this test it looks like even if the temp-table has an unique index all 3 records are visible so looks like the unique constrain does not apply so I'm at lost here :(

#12 - 03/18/2021 04:02 PM - Ovidiu Maxiniuc

Marian Edu wrote:

> Ovidiu, for temp-table I can add the 'release' for default buffer buffer in JsonExport but since this behavior is probably same in xml serialization it might be a better idea to simply flush the buffer in readAllRows instead?

No, the readAllRows() method might be (and is) used in other places. Since the code is shared in both JSON and XML serialization a single (static) method should be implemented. Ideally, the export classes should be siblings of a common abstract super class and this is the place for this method. However, these classes are quite rich and the operation might encounter some name collisions. So I think the best way is to add a static method boolean validateBuffers(TempTable tt) in serial.Util class. It would call validate / release on buffers of the temp-table as we discovered above. If it returns with success, the serialization can be performed, otherwise false is returned after eventually issuing the customized secondary error message.

> As for flushing all buffers of a temp-table (TemporaryBuffer is what we have available), the logic there is not really visible for the naked eye, too many interfaces and getting hold of the one implementation (@BufferImpl) is not always obvious.

Sorry for that but the dynamic proxy is the only way to accommodate the double interface of buffer and DMO. However, you might be right here. I do not know if a table is able to easily iterate all its buffers.
Eric, do we have such a list? I am not aware of it. Otherwise a list (LIFO) must be added to AbstractTempTable.

> The whole 'persistence' stuff is quite sensitive so it's probably we stay away from it all together, did a quick test to see how query/for each behaves and there are differences between 4GL and FWD - oo/json/quirks/table_scope.p.

I am looking at this right now.

> While in 4GL records in different temp-table's buffers can be created and have data that violates the unique index constraint doing a for each on each of those buffers will indeed flush the record and in the end the buffer is not available anymore. Now, if there is no unique index defined 4GL seems to flush the records automatically so when one open the query (for each) all records are visible. If the table has unique indexes then only the record in the respective buffer is flushed (successfully) and all others are not visible to the query. Most probably it tries to flush all buffers of the temp-table but ignore the errors if any, that's why for a table with no unique index all records are visible.

This is how FWD should also work.

> However, if flushing the buffer used in the query/for each raise error that is thrown back and the buffer remains with the invalid/uncommitted record.
> No idea how for each works in FWD but for this test it looks like even if the temp-table has an unique index all 3 records are visible so looks like the unique constrain does not apply so I'm at lost here :(

for each will release the buffer in FWD, too, because it is used to store the records fetched from database.

#13 - 03/18/2021 04:22 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Sorry for that but the dynamic proxy is the only way to accommodate the double interface of buffer and DMO. However, you might be right here. I do not know if a table is able to easily iterate all its buffers.
> Eric, do we have such a list?

I have not implemented such a list and I'm not aware of one. The closest thing I know of is that objects can register themselves as RecordChangeListeners with the ChangeBroker, and that class maintains a registry of these (but not as a simple list). Many of the listeners are buffers, but this mechanism is not really what we need for this case.

#14 - 04/01/2021 09:14 AM - Greg Shah

Is this task complete as of 3821c rev 12222?

#15 - 04/01/2021 09:15 AM - Greg Shah

- Assignee changed from Eric Faulhaber to Ovidiu Maxiniuc

#16 - 04/01/2021 11:22 AM - Ovidiu Maxiniuc

- Assignee changed from Ovidiu Maxiniuc to Eric Faulhaber

Greg Shah wrote:

> Is this task complete as of 3821c rev 12222?

Yes. Now all temporary buffers register to their parent TEMP-TABLE. The process is transparent. There is only one method exposed:

`AbstractTempTable.validateAllBuffers()`

This needs to be called at the moment of the serialization, as we discussed above. Normally, if the process ends with success, true is returned. However, it can be usually ignored as an ErrorConditionException will be thrown (and the validation process is halted) if validation fails for a buffer. Since this (Java) method is to be called from a OE method (which normally return No instead of raising an error condition), the call must be bracketed between try / catch and in the event of the ErrorConditionException being thrown, the caller should simply return false (or new logical(false)).

#17 - 04/01/2021 11:29 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from New to Test

- Assignee changed from Eric Faulhaber to Ovidiu Maxiniuc

#18 - 06/13/2021 09:16 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Greg Shah wrote:
>
> > Is this task complete as of 3821c rev 12222?
>
> Yes. Now all temporary buffers register to their parent TEMP-TABLE. The process is transparent. There is only one method exposed:
> [...]
>
> This needs to be called at the moment of the serialization, as we discussed above. Normally, if the process ends with success, true is returned. However, it can be usually ignored as an ErrorConditionException will be thrown (and the validation process is halted) if validation fails for a buffer. Since this (Java) method is to be called from a OE method (which normally return No instead of raising an error condition), the call must be bracketed between try / catch and in the event of the ErrorConditionException being thrown, the caller should simply return false (or new logical(false)).

Marian, are you already using, or do you plan to use, this AbstractTempTable.validateAllBuffers() API? I just fixed a memory leak related to this in 3821c/12530 and in doing so, I noticed it is not being called by any code, at least not in that development branch. Maybe you are using it in a branch that has not been merged back to 3821c in a while? Please let me know whether you still need it.