Database - Bug #7535

Temporary tables require trailing recid component even for unique indexes

07/13/2023 06:06 AM - Alexandru Lungu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Dănuț Filimon	**% 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 #7647: Leverage internal H2 _ROWID_ with FWD recid	**New**

History

#1 - 07/13/2023 06:10 AM - Alexandru Lungu

- Assignee set to Alexandru Lungu

- Status changed from New to WIP

My investigation for #7474 and #7455 lead me to the following:

```
def temp-table tt field f1 as int index idx as primary unique f1.

do transaction:
    create tt. tt.f1 = 1.
end.

define buffer buf for tt.
for each tt.
    message "tt:" recid(tt) tt.f1.
    do transaction: // test 1 without transaction; test 2 with transaction
       delete tt.
    end.
    create buf.
    buf.f1 = 1.
    message "bf:" recid(buf) buf.f1.
end.
```

I've done 2 tests (with and without the transaction **keyword**):

Test	4GL	FWD

WITHOUT TRANSACTION	`tt: 2004 1`
	`bf: 2005 1`
	`tt: 2005 1`
	`bf: 2004 1`
		`tt: 1 1`
		`bf: 1 1`

WITH TRANSACTION	`tt: 2004 1`
	`bf: 2004 1`
		`tt: 1 1`
		`bf: 2 1`

The results are quite opposite in FWD comparing to 4GL. In 4GL, the recid can be reclaimed only after the transaction that freed the reclaimed key ended. In the transaction test, 4GL freed the recid in the transaction and buf reclaimed it. In the without transaction test, 4GL didn't freed the recid until the end of the loop iteration. That is why buf was assigned a new recid.

In FWD, the key is reclaimed eagerly. However, if a transaction is used, the key isn't reclaimed. I double checked and I am sure the results are accurate . This is quite confusing and should be fixed at some point.

However, very important observations here are that:

- for each invalidation depends on the assigned recid. In the transaction case, for each doesn't iterate the new record because it has the same recid as the previous entry. However, in the without transaction example, the created record is iterated because it has a greater recid.
- using by f1 desc by recid desc makes the without transaction for-each iterate only once. This is because 2005 record is no in the "iteration" order of the for each, so it is missed.

With these observations, I am quite sure that we actually **need recid** as a trailing index component to unique indexes in temp records. Currently, we have:

```
         if (!isDirty && !index.isUnique())
         {
            // add [id] to make it unique:
            Property fieldId = ReservedProperty.ID;
            sb.append(", ").append(fieldId.column);

            if (lastDesc)
            {
               sb.append(" desc");
            }
         }
```

I am planning to do some skim tests without !index.isUnique() conditional. From first testing iterations, there were no regressions. I am a bit afraid however of the performance impact.

#2 - 07/13/2023 09:42 PM - Ovidiu Maxiniuc

Very good observation. This is an edge-case scenario we (or at least I) never encountered before. There are a lot of aspects to analyse here:

- the unicity of the index, as you mentioned;
- the presence of the index itself beside the option to force a specific sorting order;
- permanent tables. Do they behave in similar way?

Side question: I noticed the line for each tt. ends with a DOT, not a COLON. Was thin intentional?

#3 - 07/14/2023 03:22 AM - Alexandru Lungu

- Assignee changed from Alexandru Lungu to Dănuț Filimon

- % Done changed from 0 to 30

- the unicity of the index, as you mentioned;

In non-unique indexes, we already append a trailing recid. We can test this, especially how recid behaves, when a "sort-band" (i.e. a segment of records that are equal according to an index) is encountered.

- the presence of the index itself beside the option to force a specific sorting order;

Ovidiu, I am not quite sure what this means. Do you mean testing explicit BY clauses to force certain indexes? Or maybe force reversed indexes?

- permanent tables. Do they behave in similar way?

This is the next testing attempt.

> Side question: I noticed the line for each tt. ends with a DOT, not a COLON. Was thin intentional?

It wasn't intentional. I think I had a FIND or something that I rewrote, but missed changing the DOT into COLON. I double-checked FWD's conversion now and DOT behaves like COLON. I changed it to COLON now and I get the same results.

Danut, please make a full test suite with text output with temp tables and persistent tables. Please address this exact issue:

- reclaim / don't reclaim the recid of the deleted record
- use unique / non-unique indexe
- check with different BY clauses combinations:
  - without BY
  - BY over the indexed field (asc/desc)
  - BY over recid (asc/desc)
  - BY over the indexed field and recid (asc/asc, asc/desc, desc/asc, desc/desc)
- attempt to reclaim an lower recid: in my example, buf reclaims the deleted record's recid. However, you can make more records (e.g. with recid 1, 2, 3), when on the second record (recid 2), delete the first record with (recid 1). This way, the new buf will reclaim recid 1 and won't be iterated.
- temp / persistent tables

There is already some adjacent work on #7474; try to stick your examples to #7535 (reffering only to recid as being relevant in unique index traversal). I have the following fix already:

```
=== modified file 'src/com/goldencode/p2j/persist/TempTableHelper.java'
--- old/src/com/goldencode/p2j/persist/TempTableHelper.java     2023-06-30 10:52:48 +0000
+++ new/src/com/goldencode/p2j/persist/TempTableHelper.java     2023-07-13 10:49:17 +0000
@@ -493,7 +493,14 @@
             lastDesc = comp.isDescending();
          }

-         if (!isDirty && !index.isUnique())
+         // temp-tables require a trailing recid component to satisfy the order of the records when iterating
+         // even if unique:
+         // * FWD can store equal records that weren't nursed in the database, violating the unique index
+         // * FWD should be able to iterate indexes in dynamic mode; if a record was replaced with another:
+         // ** if the new record reclaimed the same recid, the record is skipped
+         // ** if the new record claimed a fresh recid and has the same unique data as the old record,
+         //    it is iterated only if it claimed a higher recid (or lower if defined with desc). See #7535.
+         if (!isDirty)
          {
             // add [id] to make it unique:
             Property fieldId = ReservedProperty.ID;
```

However, I am not sure it is enough :) I need a better understanding of what should be done here. Please test without this patch ad report the mismatches. Please re-test with this patch and report what was fixed.

#4 - 07/14/2023 09:10 PM - Ovidiu Maxiniuc

Alexandru Lungu wrote:

> **Ovidiu**, I am not quite sure what this means. Do you mean testing explicit BY clauses to force certain indexes? Or maybe force reversed indexes?

What I mean here is whether the lack of the index changes the outcome. That is defining the table as simple as: def temp-table tt field f1 as int.
A similar 2nd case: keep the index, but force a specific BY sorting so that the index *cannot be used* for record navigation (this table is too simple for this).

#5 - 07/16/2023 10:25 AM - Alexandru Lungu

Ovidiu Maxiniuc wrote:

> A similar 2nd case: keep the index, but force a specific BY sorting so that the index *cannot be used* for record navigation (this table is too simple for this).

Won't this force a PreselectQuery in the first place? In this case, there is no record invalidation.

#6 - 07/17/2023 03:04 PM - Ovidiu Maxiniuc

- % Done changed from 30 to 0

- Assignee changed from Dănuț Filimon to Alexandru Lungu

In fact, I was thinking first of 4GL side, to see whether strange issues occur.

#7 - 07/18/2023 08:22 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, please make a full test suite with text output with temp tables and persistent tables. ...

Committed testcases/rev.2381. Added tests for recid reclaiming based on the specifications mentioned in #7535-3. I wrote the test set earlier, but eventually managed to simplify it into 3 test files.

#8 - 07/20/2023 03:08 AM - Alexandru Lungu

- % Done changed from 0 to 50

Danut, do you have any feedback on the change in #7535-3? According to your tests:

- Are there problems that occurs without the patch are solved by the patch?
- Are there any tests that are "broken" by the patch?
- Can you test a customer application with the patch and minimally ensure there is no obvious regression?

Double-check you meet Ovidiu's requirements from #7535-2, #7535-4, #7535-6.
If everything is fine, I will test this patch for regressions / performance myself and consider merging.

#9 - 07/20/2023 08:59 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, do you have any feedback on the change in #7535-3? According to your tests:

I tested with trunk/rev.14654. There were a few cases where the FWD code would run in an infinite loop, after ruling out the tests that caused this issue and applied the patch, new instances of infinite loops appear. I downgraded the fwd-h2 version from revision 24 to 22 as you suggested and still have infinite loops running.

Committed tetcases/rev.2382. I adjusted the recid reclaiming tests since I managed to copy-paste the wrong table/test in a few cases and added an infinite loop check that will leave the FOR EACH loop if it executed too many times. The current test-set works without problems in 4GL, even if the output of the code is **no record available**.

Here is an overview of the tests that ran into an infinite loop:

Test	fwd-22, trunk	fwd-22, patched trunk	fwd-24, trunk	fwd-24. patched trunk
test01.i - tt01	normal	infinite	normal	infinite
test01.i - tt02	infinite	infinite	infinite	infinite
test01.i - tt01 BY f1	normal	infinite	normal	infinite
test01.i - tt02 BY f1	infinite	infinite	infinite	infinite
test01.i - tt01 BY f1 BY RECID	normal	infinite	normal	infinite
test01.i - tt01 BY f1 BY RECID DESC	normal	infinite	normal	infinite
test01.i - tt01 BY f1 DESC	normal	normal	normal	infinite
test01.i - tt02 BY f1 DESC	normal	normal	infinite	infinite
test01.i - pt2	infinite	infinite	infinite	infinite
test02.i - pt2	infinite	infinite	infinite	infinite
test01.i - pt2 BY f1	infinite	infinite	infinite	infinite
test02.i - pt2 BY f1	infinite	infinite	infinite	infinite

Between fwd-h2 revision 22 and 24, there is a single case that is broken, while the rest that previously worked are not stuck into a loop. I don't think the current patch will work properly.
Note that all the tests that were not mentioned in the table above work in all 4 cases.

> Can you test a customer application with the patch and minimally ensure there is no obvious regression?

Should I test it considering what was mentioned above?

EDIT: Mentioned the wrong commit for testcases. It is rev.2382.

#10 - 07/20/2023 10:50 AM - Alexandru Lungu

- Assignee changed from Alexandru Lungu to Dănuț Filimon

I wonder how many of these tests fail due to "bad recid reclaim" vs "bad LAZY iteration" not considering recid. I suggest fixing #7535-1 first, so proper recid is reclaimed. We will redo the tests then.

#11 - 07/21/2023 02:42 AM - Alexandru Lungu

Danut, can you extend the results from #7535-9 with the patch from #7373-35 (FWD-H2 1.22 and FWD-H2 1.24)?

#12 - 07/21/2023 02:43 AM - Alexandru Lungu

```
Index: src/com/goldencode/p2j/persist/RecordBuffer.java
--- a/src/com/goldencode/p2j/persist/RecordBuffer.java
+++ b/src/com/goldencode/p2j/persist/RecordBuffer.java
@@ -3329,7 +3329,7 @@
             {
                // in batch assign mode, standard validation happens before assign triggers are fired
                Record dmo = buffer.getCurrentRecord();
-               buffer.validateMaybeFlush(dmo, false, false);
+               buffer.validateMaybeFlush(dmo, buffer.isTemporary(), false);

                // fire assign triggers, if any
                TriggerTracker tracker = buffer.getTriggerTracker();
```

For reference, this is the change.

#13 - 07/21/2023 03:31 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, can you extend the results from #7535-9 with the patch from #7373-35 (FWD-H2 1.22 and FWD-H2 1.24)?

I used the patch from from #7535-12 and got the following results:

Test	fwd-22, trunk	fwd-22, patched trunk	fwd-24, trunk	fwd-24, patched trunk

test01.i - tt02	infinite	infinite	infinite	infinite
test01.i - tt02 BY f1	infinite	infinite	infinite	infinite
test01.i - tt02 BY f1 DESC	normal	normal	infinite	infinite
test01.i - pt2	infinite	infinite	infinite	infinite
test02.i - pt2	infinite	infinite	infinite	infinite
test01.i - pt2 BY f1	infinite	infinite	infinite	infinite
test02.i - pt2 BY f1	infinite	infinite	infinite	infinite

When comparing the results using the new patch with the results obtained in #7535-9, the only thing that is noticeable is that there are less broken entries and the results of the current patch match the previous ones obtained.

Note: The tests were done using only the #7535-12 patch, trunk/rev.14654.

#14 - 07/21/2023 03:38 AM - Alexandru Lungu

The patch was targeting only temp database, so it was clear that pt tests won't be improved. Anyway, this means there are 3 tests left to handle. Can you do a quick check with combined patches for these 3 tests (only fwd-22)?

#15 - 07/21/2023 03:53 AM - Dănuț Filimon

Alexandru Lungu wrote:

> The patch was targeting only temp database, so it was clear that pt tests won't be improved. Anyway, this means there are 3 tests left to handle. Can you do a quick check with combined patches for these 3 tests (only fwd-22)?

More tests are failing when also applying the #7535-3 patch.

Test	fwd-22, trunk (2 patches)
test01.i - tt01	infinite
test01.i - tt02	infinite
test01.i - tt01 BY f1	infinite
test01.i - tt02 BY f1	infinite
test01.i - tt01 BY f1 BY RECID	infinite
test01.i - tt01 BY f1 BY RECID DESC	infinite

The only exception seems to be test01.i - tt02 BY f1 DESC, which is solved now.

#16 - 07/21/2023 04:03 AM - Alexandru Lungu

I think it is time for some digging :) As I said previously, lets start with the recid reclaiming process.

#17 - 07/24/2023 09:06 AM - Dănuț Filimon

After investigating how the recid is reclaimed for temporary tables, I made a few changes to allow reclaiming only when a transaction is in use.

I also arrived to the conclusion that the patch mentioned in #7535-3 is necessary in order to iterate over the new record created (with a new recid) and eventually reclaim the recid of the record that was deleted, but adding the patch results into an infinite loop for the #7535-1 test. Even if the infinite loop problem is solved, we still need to make it possible to reclaim the recid after each loop iteration.

#18 - 07/26/2023 04:12 AM - Alexandru Lungu

Lets make baby steps here. We need the recid reclaim be sound first. At that point, we shall see how we can continue with the infinite loops.

I created 7535a. Please commit your recid fix there from #7535-17. If the fix is not that large (<10 lines), please post it here as well for easier reference.

#19 - 07/26/2023 04:29 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Lets make baby steps here. We need the recid reclaim be sound first. At that point, we shall see how we can continue with the infinite loops.
>
> I created 7535a. Please commit your recid fix there from #7535-17. If the fix is not that large (<10 lines), please post it here as well for easier reference.

I've done the following change for temporary tables in order to get the results from #7535-17.

```
=== modified file 'src/com/goldencode/p2j/persist/TemporaryBuffer.java'
--- old/src/com/goldencode/p2j/persist/TemporaryBuffer.java    2023-06-27 14:03:15 +0000
+++ new/src/com/goldencode/p2j/persist/TemporaryBuffer.java    2023-07-26 08:13:41 +0000
@@ -7382,7 +7382,7 @@

       try
       {
-         if (autoCommit || loopingDelete)
+         if (!autoCommit || loopingDelete)
          {
             id = getCurrentRecord().primaryKey();
          }
@@ -7464,11 +7464,11 @@
             }
          }

-         if (autoCommit || loopingDelete)
+         if (!autoCommit || loopingDelete)
          {
             local.reclaimKey(getDMOInterface(), getMultiplexID(), id);

-            if (autoCommit)
+            if (!autoCommit)
             {
                local.commit(true);
             }
```

#20 - 07/27/2023 08:21 AM - Alexandru Lungu

Danut, I don't think the changes are quite right. autoCommit means:

- "yes": there is no active transaction (do transaction), so the changes should take effect immediately. This is fine (as commit should be called), but the reclaiming itself shouldn't happen immediately. I don't actually know when it should (maybe it won't be ever reclaimed). **Please refactor the code in #7535-1 and identify where 2004 recid is reclaimed if there is no transaction.** Out-side for-each? Out-side the procedure? Is there any documentation on this?
- "no": there is a current application transaction, so we will do the reclaiming then. This should work with the current code. However, TemporaryBuffer.reclaimKeys seems to be no longer used. Please check other Redmine tasks that may have eliminated reclaimKeys. It should be called when a transaction ends, right? **Anyway, please check that inside the transaction, the reclaiming is not possible - but only after the transaction ends. Also check sub-transactions.**

#21 - 07/28/2023 07:35 AM - Dănuț Filimon

Alexandru Lungu wrote:

> **Please refactor the code in #7535-1 and identify where 2004 recid is reclaimed if there is no transaction.** Out-side for-each? Out-side the procedure? Is there any documentation on this?

After testing different cases with/without transactions, procedures, here's a summary of my findings on how the recid is reclaimed:

- Deleting a record will make it's recid available immediately after a transaction/iteration of a block. The recid of a record deleted in a FOR block can be reclaimed outside of the block (after all records are deleted) or in the next iteration (deleted record in iteration 1, reclaimed recid in iteration 2).
- If you use a separate transaction to delete a record in a FOR block, then the recid can be reclaimed at the end of the transaction, still inside the block.
- Recid can be reclaimed inside/outside of a procedure.

There is no documentation on the reclaim process, just a few unrelated forum posts/articles that mention that it is possible but do not offer a lot of context.

> Please check other Redmine tasks that may have eliminated reclaimKeys.

I found no tasks that mention reclaimKeys being removed, but there were a lot of changes done in trunk/rev.11348 when Hibernate was replaced. There were two methods that used reclaimKeys at that time, TemporaryBuffer.commit() (method was commented) and RecordBuffer.rollback() (method was mostly commented). I do not know yet why their functionality was removed. Since they were commented it's probably because it was meant fix them at some point (?). I see that RecordBuffer.rollback() was changed, but the method from TemporaryBuffer was left as it is (or was replaced by Context.commit()).

#22 - 07/28/2023 07:57 AM - Alexandru Lungu

> TemporaryBuffer.commit() (method was commented) and RecordBuffer.rollback() (method was mostly commented)

Note that TemporaryBuffer.Context implements Commitable and there is the whole reclaim logic already. The same goes for rollback. Are TemporaryBuffer.Context.commit and @TemporaryBuffer.Context.rollback still called properly in our code?

#23 - 07/28/2023 08:14 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Note that TemporaryBuffer.Context implements Commitable and there is the whole reclaim logic already. The same goes for rollback. Are TemporaryBuffer.Context.commit and @TemporaryBuffer.Context.rollback still called properly in our code?

I only did tests which called TemporaryBuffer.Context.commit and didn't make any that force a rollback. But I see that TemporaryBuffer.Context.rollback is used in TxWrapper and TransactionManager in a similar manner as the commit method.

#24 - 07/28/2023 08:37 AM - Alexandru Lungu

So what is the problem then? If TemporaryBuffer.Context.commit is called:

- is it called at bad times? too late or too soon?
- is the code there broken?

#25 - 07/28/2023 09:22 AM - Dănuț Filimon

Alexandru Lungu wrote:

> So what is the problem then? If TemporaryBuffer.Context.commit is called:
>
> - is it called at bad times? too late or too soon?
> - is the code there broken?

At the end of each iteration, the keys are not reclaimed for temporary tables. RecordBuffer.iterate() should handle this, but it only does so for permanent tables.

#26 - 07/28/2023 10:23 AM - Dănuț Filimon

- File 7535-20230728.patch added

Keys should be reclaimed at the end of each iteration and when the block finishes, they should not be reclaimed too early (when the record is deleted). I attached a patch for the current issue, which shows correct results for the #7535-1 example. It needs to be tested, so I didn't commit these changes yet.

#27 - 08/04/2023 06:44 AM - Alexandru Lungu

- Related to Bug #7647: Leverage internal H2 _ROWID_ with FWD recid added

#28 - 08/09/2023 04:56 AM - Dănuț Filimon

Using the patch posted in #7535-26, I've done similar tests to #7535-15 on 7535a using FWD-22 and all the previous patches available (#7535-12 and #7535-3):

Test	7535-26 patch	7355-26 & 12	7535-26 & 3	7535-26 & 3 & 12
test02.i - tt01	normal	normal	infinite	infinite
test02.i - tt02	infinite	infinite	infinite	infinite
test02.i - tt01 BY f1	normal	normal	infinite	infinite
test02.i - tt02 BY f1	infinite	infinite	infinite	infinite
test02.i - tt01 BY f1 BY RECID	normal	normal	infinite	infinite
test02.i - tt01 BY f1 BY RECID DESC	normal	normal	infinite	infinite

Previously, all tests on test02.i with temporary tables passed. I am currently looking into the two failing tests.

#29 - 08/11/2023 05:27 AM - Dănuț Filimon

Committed 7535a/rev.14672. Applied the patch from #7535-26 after making another small change.

- Added reclaimKeys method to TemporaryBuffer.
- When deleting a record using TemporaryBuffer.delete(), the id should always be added to pendingReclaimedKeys. If the delete action is done in a transaction, the id should be reclaimed immediately.
- Keys are reclaimed at the end of each iteration or when the block ends.

I looked into the failing tests from #7535-28 and the problem is that the ids are added whenever TemporaryBuffer.delete() is called but are not actually reclaimed at all. After running both of the failing tests separately, there was no issue and both finished with the results expected. I assume that previous tests somehow influence the execution of future tests which should not happen.

The keys are not reclaimed in the for each block since there are multiple implementations for iterate and finished available (e.g. PreselectQuery.cleaner object and PreselectQuery iterate and finished methods are called).

Another factor that may contribute to infinite loop is the optimization done to a FOR EACH ... DELETE ... which is converted to deleteAll. At the end of the loopDelete and removeRecords the ReclaimTable is simply discarded with no actual pending key being reclaimed.

#30 - 08/16/2023 03:44 AM - Dănuț Filimon

I mentioned that keys are not reclaimed in simple FOR EACH blocks and after making the necessary changes to reclaim keys of the PreselectQuery components using the iterate, finished methods and cleaner object, this change worked properly for all temporary table tests, with no infinite loops.

I created a python script to compare the results between 4GL and FWD of the test suite because there is too much data to compare and the different values used for the recid are too varying (the recid starts at a different number in 4GL, while in FWD it starts at 1). At the moment, I see problems with a mismatch between the number of iterations in **8** tests and wrong outputs in **6** tests. All the **6** tests use **RECID DESC**, but the FWD results are actually in ascending order which leads me to think that there is another problem happening here.

#31 - 08/16/2023 06:52 AM - Dănuț Filimon

When running the test01.i with tt01 BY RECID DESC test, I end up with this select statement:

```
select tt01 from Tt01_1_1__Impl__ as tt01 where (tt01._multiplex = ?0)  order by tt01._multiplex asc, tt01.f1
asc, tt01.recid asc
```

Even if RECID DESC is specified, the sort condition is different. I think the problem is caused by Presorter.addSortCriterion which is directly called in the converted code (query0.addSortCriterion(((RecidExpr) () -> tt01.recordID()), true);) and ends up negating the value provided when creating the SortHelper. Currently I can't test this assumption because of an ongoing issue with **jboss** repository resource which stops the project from compiling (I tried to reorder the repositories but it didn't work).

#32 - 08/17/2023 03:20 AM - Alexandru Lungu

Danut, regarding the recid mismatch, please refer to #7647-2. Use Ovidiu's patch just to clear up **some** mismatches. Anyway, I will assign you to #7647, but don't mark it as top priority for now. You are the closest to the temp-tables recid generation, so your expertise here might come in handy with #7647.

#33 - 08/17/2023 03:23 AM - Alexandru Lungu

> Currently I can't test this assumption because of an ongoing issue with jboss repository resource which stops the project from compiling (I tried to reorder the repositories but it didn't work).

This is something that affects us periodically. Try moving the jboss repository as the **last** repository in build.gradle. This will prioritize the look-up in GCD private repository. We may want to actually commit such change. AFAIK, we are facing a load balancer that moves us sometimes to a faulty resource (which returns 400 instead of 404 and makes Gradle look-up crash).

#34 - 08/17/2023 03:46 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, regarding the recid mismatch, please refer to #7647-2. Use Ovidiu's patch just to clear up **some** mismatches. Anyway, I will assign you to #7647, but don't mark it as top priority for now. You are the closest to the temp-tables recid generation, so your expertise here might come in handy with #7647.

Thank you, I already completed and improved my script to clear these mismatches from the results so it isn't that much of a problem currently, but I will use the patch if I will need to run the full test suite again.

> This is something that affects us periodically. Try moving the jboss repository as the last repository in build.gradle.

Already done so, I tried reordering the repositories but it still didn't work. The problem was eventually solved by the source later in the day.

#35 - 08/17/2023 03:58 AM - Dănuț Filimon

Committed 7535a/rev.14673. This is a necessary change that I mentioned in #7535-29 about the PrelectQuery.cleaner object and using it to reclaim keys in simple for each blocks that are not within a transaction. This change solved all of the infinite loops for the temporary table tests and allowed me to review the results of the tests, which I explained a bit in #7535-30.

One very interesting case among the failing tests is the following:

- 6 records are initially created for a temporary table within a transaction.
- In a FOR EACH block on the temporary table, we save the current records value in a variable, then we deleted it (no transaction). We then create a new record with the saved value using a buffer.
- At the end we delete all the records.

We expect to reclaim the recid at the end of the iteration and use it when creating a new record with the buffer in the next iteration which it's actually the current correct scenario, but the problem is that the next record that the FOR EACH **iterates** is the **newly created record** instead of the **next available one**.

#36 - 08/17/2023 04:20 AM - Dănuț Filimon

I also found CompoundQuery.cleaner which is also a Finalizable instance, it's a similar situation to PreselectQuery and this should also allow key reclaiming. I will wait for a review on #7535-29 and #7535-35 and make the change after.

#37 - 08/17/2023 07:07 AM - Alexandru Lungu

Danut, can you paste here the testcase which causes the issue in #7535-35? Thank you.

#38 - 08/17/2023 07:37 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, can you paste here the testcase which causes the issue in #7535-35? Thank you.

The test is similar to the one from #7535-1, with the exception being iOldValue which is used to create a record with the previous value that was deleted. I had to change it a bit to reduce it's size, but it's showing the same results as before.

```
DEFINE TEMP-TABLE tt01 FIELD f1 AS INTEGER INDEX idx1 AS PRIMARY UNIQUE f1.
DEFINE BUFFER buf01 FOR tt01.
DEFINE VARIABLE iIter AS INTEGER NO-UNDO.
DEFINE VARIABLE iOldValue AS INTEGER NO-UNDO.

DO TRANSACTION:
    DO iIter = 1 TO 6:
        CREATE tt01. tt01.f1 = iIter.
    END.
END.

FOR EACH tt01.
    MESSAGE "tt:" RECID(tt01) tt01.f1.
    iOldValue = tt01.f1.
    DELETE tt01.
    CREATE buf01.
    buf01.f1 = iOldValue.
    MESSAGE "buf:" RECID(buf01) buf01.f1.
END.
```

#39 - 08/17/2023 12:09 PM - Eric Faulhaber

Dănuț Filimon wrote:

> When running the test01.i with tt01 BY RECID DESC test, I end up with this select statement:
>
> ```
> select tt01 from Tt01_1_1__Impl__ as tt01 where (tt01._multiplex = ?0)  order by tt01._multiplex asc, tt01
> .f1 asc, tt01.recid asc
> ```

> Even if RECID DESC is specified, the sort condition is different. I think the problem is caused by Presorter.addSortCriterion which is directly called in the converted code (query0.addSortCriterion(((RecidExpr) () -> tt01.recordID()), true);) and ends up negating the value provided when creating the SortHelper. Currently I can't test this assumption because of an ongoing issue with **jboss** repository resource which stops the project from compiling (I tried to reorder the repositories but it didn't work).

I wouldn't expect BY RECID DESC to work. That doesn't look like valid 4GL code.

That is, I don't understand the use of RECID in the context of a BY clause. Is that a field name? RECID is a 4GL keyword, which is not allowed as a field name. I tried defining a temp-table with that as the name of a field and it causes compiler errors. I had never tried using RECID in a BY clause, so, thinking BY RECID DESC maybe was an undiscovered (by me, at least) shorthand for BY RECID(tt01) DESC, I also tried using BY RECID DESC with a FOR EACH statement. That was rejected, too.

Please help me understand what the test case is doing here, and how you are (legally) referencing RECID as a field name.

#40 - 08/18/2023 02:28 AM - Dănuț Filimon

Eric Faulhaber wrote:

> I wouldn't expect BY RECID DESC to work. That doesn't look like valid 4GL code.
>
> That is, I don't understand the use of RECID in the context of a BY clause. Is that a field name? RECID is a 4GL keyword, which is not allowed as a field name. I tried defining a temp-table with that as the name of a field and it causes compiler errors. I had never tried using RECID in a BY clause, so, thinking BY RECID DESC maybe was an undiscovered (by me, at least) shorthand for BY RECID(tt01) DESC, I also tried using BY RECID DESC with a FOR EACH statement. That was rejected, too.
>
> Please help me understand what the test case is doing here, and how you are (legally) referencing RECID as a field name.

Sorry for the misunderstanding, I shortened the terms when writing previous data tables (sometimes they can be really extensive and can take a lot of space). Indeed, the BY RECID DESC is a short version of BY RECID(tt01) DESC.

The test case is similar to the one mentioned in #7535-38 with the following 2 exceptions:

- FOR EACH tt01. should be FOR EACH tt01 BY RECID(tt01) DESC.
- DELETE tt01. should be done in a transaction:

```
DO TRANSACTION:
    DELETE tt01.
END.
```

The problem with the test is that in FWD, the results are displayed in ascending order, while the results in 4GL are displayed in descending order.

#41 - 08/18/2023 04:24 AM - Alexandru Lungu

Review of 7535a

- Doing a reclaim on iterate looks right. However, I think iterate is called only for the buffers that are used by the for each (or other repeating block). What happens to the other buffers (on other tables / other multiplex)? What if we iterate with tt, but create/delete records for a tt2 that is not part of any record-phrase in the for-each. Is the iterate method called for tt2? Are the keys reclaimed for tt2 at the end of each for-each loop?
- I guess it is OK to add reclaimKeys to finished method. However, note the existence of a deleted method as well. AFAIK, this is called for dynamic buffers that are deleted at some point using DELETE OBJECT. Maybe you should move reclaimKeys in finishedImpl? My concern here is that this is not dependent upon the transactional context. A buffer can get out-of-scope while a transaction is still active (procedure A calls procedure B in a transaction, procedure B has a record buffer over which it executed DELETE, you return to A and create a new record, will it reclaim the key just because the record buffer in B got out-of-scope?). Also, a buffer can be deleted while a transaction is still active - I am not sure this will trigger reclaiming. **Please test this asap.**
- The changes in PreselectQuery are quite invasive. There are 4 new reclaimKeys calls for different scenarios. They look logical to me, as these trigger the reclaim for its underlying components. However, this is not transactional related in any way. What if the iteration happens in a larger transaction block? What if the for each body is **not** transactional (all operations are done in nested do transaction blocks)?
- Rename reclaimKey to pendReclaimKey as it is quite confusing in regard to reclaimKeys that actually reclaims the keys.

Please take a look on RecordBuffer.iterate where reclaimKeys is called. It is called only when !bufferManager.isTransaction(), so it is quite related to the transactional context. Also, there is currentScope == activeScopeDepth. I think this is used in for each tt: for each tt2: [...] cases (nested loops). Please extend your test set with such nested loops. Also attempt something like for each tt: for each btt: [...], where tt and btt are buffers over the same table.

Important!

It is nice to have a large testing suite working, but I think it is biased by the example in #7535-1 which is quite trivial (with not that many transactions, nesting, external procedures, etc.). Also, please check this feature in regard to UNDO/NO-UNDO properties and if the UNDO operations reverts the pendingKeys / reclaiming process.

If the temp-table is no-undo, will it do eager reclaiming? If yes, maybe we need to make sure we don't ever reclaim keys while the undoable was not yet committed / roll-backed. Danut, please check if TransactionManager.commit and TransactionManager.rollback are better to be used. My point is that we may have been mislead by Finalizable, but we actually needed Commitable. So instead of working with iterate and finalize, we should have worked with commit and rollback. Please check if this is the case. If yes, I will create a 7535b for that matter.

#42 - 08/18/2023 06:48 AM - Dănuț Filimon

Alexandru Lungu wrote:

> What happens to the other buffers (on other tables / other multiplex)? What if we iterate with tt, but create/delete records for a tt2 that is not part of any record-phrase in the for-each. Is the iterate method called for tt2? Are the keys reclaimed for tt2 at the end of each for-each loop?

In **4GL**, the recid is reclaimed at the end of the loop.

In **FWD**, the recid is also reclaimed even if another table is iterated. This is because the keys that should be reclaimed belong to the same context.

> My concern here is that this is not dependent upon the transactional context. A buffer can get out-of-scope while a transaction is still active (procedure A calls procedure B in a transaction, procedure B has a record buffer over which it executed DELETE, you return to A and create a new record, will it reclaim the key just because the record buffer in B got out-of-scope?).

In **4GL**, the recid of the deleted record (which was deleted in procedure B), will be immediately available when creating the new record, be it creating it normally or through another procedure. It is also notable that the recid is reclaimed even if the procedure B is not within a transaction. However, **FWD** will reclaim the key only at the end of the iteration or when the procedure B is executed inside a transaction.

> Also, a buffer can be deleted while a transaction is still active - I am not sure this will trigger reclaiming. Please test this asap.

Can you please elaborate?

> The changes in PreselectQuery are quite invasive. There are 4 new reclaimKeys calls for different scenarios. They look logical to me, as these trigger the reclaim for its underlying components. However, this is not transactional related in any way. What if the iteration happens in a larger transaction block? What if the for each body is not transactional (all operations are done in nested do transaction blocks)?

Yes, the changes to PreselectQuery are not transactional related. The only thing I could notice is that between a transaction and a non-transaction is that TransactionManager.processFinalizables calls the iterate/finished methods of the buffer when a transaction is involved. But even if the for each body is not transactional, isn't it considered to be an implicit transaction when it directly updates the database?

I will look into expanding the test suite and experimenting with UNDO/NO-UNDO.

#43 - 08/18/2023 07:22 AM - Alexandru Lungu

Dănuț Filimon wrote:

> Can you please elaborate?

Create a dynamic buffer with CREATE BUFFER buf. Do similar work you've done in your tests. Execute DELETE OBJECT buf. At this point, the buffer is deleted and the keys should be reclaimed? Does it matter if you do the delete in a transaction or not?

> But even if the for each body is not transactional, isn't it considered to be an implicit transaction when it directly updates the database?

It is. But as long as the database updates are done in nested transaction blocks (do transaction), the for-each body won't be transactional. See:

```
for each tt:
do tranaction:
    <stuff...>
end.
message tt.recid. // this doesn't belong to an explicit 4GL transaction
end.
```

> I will look into expanding the test suite and experimenting with UNDO/NO-UNDO.

Also please check if the query is influenced by its transactional context:

```
do transaction:
   for each tt:
   do:
      <stuff...>
   end.
   message tt.recid. 
   end. // the reclaim will happen here
end. // or here?
```

#44 - 08/21/2023 07:36 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Create a dynamic buffer with CREATE BUFFER buf. Do similar work you've done in your tests. Execute DELETE OBJECT buf. At this point, the buffer is deleted and the keys should be reclaimed? Does it matter if you do the delete in a transaction or not?

I did 3 tests where records are deleted and then created in a loop (first test - no delete object buf, second test - delete object buf, third test - delete object buf in a transaction). The recid of the deleted record is reclaimed at the end of the iteration in the first two tests, but in the last one where the buf object is deleted in a transaction, the recid is **never reclaimed**.

> Also please check if the query is influenced by its transactional context:
>
> ```
> do transaction:
>    for each tt:
>    do:
>       <stuff...>
>    end.
>    message tt.recid.
>    end. // the reclaim will happen here
> end. // or here?
> ```

The reclaim happens at the end of the **outermost** DO TRANSACTION block.

All of my tests were done using UNDO tables, after modifying and running existent tests with NO-UNDO tables I found out that any recid that is made available is **reclaimed immediately**.

#45 - 08/21/2023 07:40 AM - Alexandru Lungu

Dănuț Filimon wrote:

> I did 3 tests where records are deleted and then created in a loop (first test - no delete object buf, second test - delete object buf, third test - delete object buf in a transaction). The recid of the deleted record is reclaimed at the end of the iteration in the first two tests, but in the last one where the buf object is deleted in a transaction, the recid is **never reclaimed**.

Can you post the third example here to discuss why the recid is never reclaimed? Mayb is because you scope the full-transaction to the external procedure so the reclaim is actually done after the procedure ends?

> The reclaim happens at the end of the **outermost** DO TRANSACTION block.

This really looks like the reclaiming is when the FULL TRANSACTION (outermost transaction block) is finished. This is a motivation to migrate to Commitable instead of Finalizable.

> All of my tests were done using UNDO tables, after modifying and running existent tests with NO-UNDO tables I found out that any recid that is made available is **reclaimed immediately**.

This should be considered carefully. It makes sense as the NO-UNDO tables are not transaction dependent, so they can do their stuff immediately. This should be implemented accordingly (reclaim immediately if the table is NO-UNDO).

#46 - 08/21/2023 07:48 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Can you post the third example here to discuss why the recid is never reclaimed? Mayb is because you scope the full-transaction to the external procedure so the reclaim is actually done after the procedure ends?

Of course.

```
DEFINE TEMP-TABLE tt01 FIELD f1 AS INTEGER INDEX idx1 AS PRIMARY UNIQUE f1.
DEFINE VARIABLE buf01 AS HANDLE NO-UNDO.
DEFINE VARIABLE iIter AS INTEGER NO-UNDO.
DEFINE VARIABLE iOldValue AS INTEGER NO-UNDO.

DO TRANSACTION:
    DO iIter = 1 TO 6:
        CREATE tt01. tt01.f1 = iIter.
    END.
END.

FOR EACH tt01.
    CREATE BUFFER buf01 FOR TABLE "tt01".
    MESSAGE "tt:" RECID(tt01) tt01.f1.
    iOldValue = tt01.f1 - 1.
    DELETE tt01.
    buf01:BUFFER-CREATE().
    buf01:BUFFER-FIELD("f1"):BUFFER-VALUE() = iOldValue.
    MESSAGE "buf:" buf01:RECID buf01:BUFFER-FIELD("f1"):BUFFER-VALUE().
    DO TRANSACTION:
        DELETE OBJECT buf01.
    END.
END.
```

If you remove the DO TRANSACTION block and keep the DELETE OBJECT buf01, recids will be reclaimed.

> This really looks like the reclaiming is when the FULL TRANSACTION (outermost transaction block) is finished. This is a motivation to migrate to Commitable instead of Finalizable.

Should the recent changes to iterate and finished be reverted? I am talking about 7535a/rev.14698 which only centered around using Finalizable.

#47 - 08/21/2023 07:56 AM - Alexandru Lungu

What about a 7535b?

#48 - 08/21/2023 08:01 AM - Dănuț Filimon

Alexandru Lungu wrote:

> What about a 7535b?

Yes, please create 7535b so that I can start working on Commitable.

#49 - 08/22/2023 04:10 AM - Dănuț Filimon

- File infinite-20230822.p added

I retested after making a change to Commitable, but I am still left with two tests that end up in an infinite loop. Previously I mentioned that this loop is caused by the execution of precedent tests, but the attached test case contradicts this idea. There is no reclaim happening in the TransactionType.SUB, neither after forcefully leaving, or caused by the FOR EACH. The Finalizable approach allowed reclaiming at the end of each iteration/block, but it was not transactional related and Commitable doesn't seem useful in this case.

#50 - 08/23/2023 10:07 AM - Alexandru Lungu

The test-cases you found are quite interesting. I've done some testing myself with nested transactions.

In 95% of the cases, the reclaiming of a key is done when the full-transaction that enclosed the delete ends. This can be either: a DO TRANSACTION, a REPEAT TRANSACTION, a FOR EACH TRANSACTION (maybe other blocks as well?). As a rule of thumb, you can reclaim keys only in scopes that can't undo the delete. This is good as the state transitioning of the key reclaiming was going to be confusing anyway.

In your example from #7535-49, I expect that the records are committed after each for-each loop. This way, the keys are made reclaimable as you can't undo the process anymore. Please write a test similar to the one in #7535-49, but with persistent tables. Add a WRITE TRIGGER and DELETE TRIGGER to the persistent table. AFAIK, these triggers should be fired only at the transaction end. This can help us confirm the moment when the transactions actually end.

Your example from #7535-46 falls into my 5% tests that look more like quirks. Why is the transaction management any different if you chose to use an inner DO TRANSACTION instead of DO - why is the parent block affected?
Constantin, Ovidiu, Eric, can you shed some light on this? It seems like the FOR EACH block behaves differently (transactional-wise) if the inner DO block is transactional or not. Why is the outer FOR-EACH transaction affected?

- It looks to me like the entire external procedure becomes a transaction if the FOR EACH nests an explicit DO TRANSACTION.
- If the FOR EACH nests only a simple DO, the FOR EACH loops are transactions.

#51 - 08/23/2023 10:23 AM - Constantin Asofiei

Alexandru, you can use COMPILE test.p LISTING test.lst to check the transaction behavior in 4GL. Keep in mind that a simple DO block has no tx properties (like SUB-TRANSACTION). Only DO TRANSACTION has.

#52 - 08/23/2023 03:59 PM - Tijs Wickardt

Constantin Asofiei wrote:

> Alexandru, you can use COMPILE test.p LISTING test.lst to check the transaction behavior in 4GL. Keep in mind that a simple DO block has no tx properties (like SUB-TRANSACTION).

COMPILE test.p SAVE=no LISTING test.lst is what I normally use, it prevents a loose .r file which can cause confusion.
It is the only proper way to see the compile time transactions and subtransactions.

> Only DO TRANSACTION has.

This isn't entirely correct, you can get implicit transaction blocks. Even without the DO statement, for example a FOR block or PROCEDURE block will be an implicit tx block if a non dynamic database CRUD action is inside.
Follow the LISTING.

You can check the runtime TRANSACTION as well (same keyword, but used as a function instead of a block modifier), which can have a bigger life span than the compile time transaction blocks.
Use DBTASKID(db) at runtime to see the OEDB transaction ID of the current transaction, if any.

#53 - 08/24/2023 03:42 AM - Alexandru Lungu

I was using COMPILE test.p LISTING test.lst at some point, but this is informative only when working with a persistent database. Doing inserts/updates/deletes on the _temp database doesn't trigger any changes in test.lst. Unless you explicitly use TRANSACTIONAL somewhere, all blocks are NO TRANS, even if the scenario creates/updates/deletes temp records.

Also, DBTASKID(db) is available only for a persistent database.

My primary concern here is that _temp is behaving differently from a persistent database:

Example	Output	Description

`DEFINE TEMP-TABLE tt01 FIELD f1 AS INTEGER INDEX idx1 AS PRIMARY f1.`
`DEFINE BUFFER buf FOR tt01.`
 	
`DO transaction:`
`    CREATE tt01. tt01.f1 = 1.`
`    CREATE tt01. tt01.f1 = 2.`
`END.`
	`tt: 2304 1`
	`buf: 2306 0`
	`transaction`
	`tt: 2305 2`
	`buf: 2307 0`
	`transaction`
		the 2304 key is **not** reclaimed by the second iteration
		this time the inner DO block is transactional

`FOR EACH tt01:`
`    MESSAGE "tt: " RECID(tt01) tt01.f1.`
`    DELETE tt01.`
`    CREATE buf.`
`    MESSAGE "buf:" RECID(buf) buf.f1.`
`    DO TRANSACTION:`
`        MESSAGE "transaction".`
`    END.`
`END.`		

`DEFINE TEMP-TABLE tt01 FIELD f1 AS INTEGER INDEX idx1 AS PRIMARY f1..`
`DEFINE BUFFER buf FOR tt01.`
 	
`DO transaction:`
`    CREATE tt01. tt01.f1 = 1.`
`    CREATE tt01. tt01.f1 = 2.`
`END.`
 	
`FOR EACH tt01:`
`    MESSAGE "tt: " RECID(tt01) tt01.f1.`
`    DELETE tt01.`
`    CREATE buf.`
`    MESSAGE "buf:" RECID(buf) buf.f1.`
`    DO:`
`        MESSAGE "transaction".`
`    END.`
`END.`
	`tt: 2304 1`
	`buf: 2306 0`
	`transaction`
	`tt: 2305 2`
	`buf: 2304 0`
	`transaction`
		the 2304 key **is** reclaimed by the second iteration
		this time the inner DO block is **not** transactional

`DEFINE BUFFER buf FOR Simple. // persistent`
 	
`FOR EACH Simple:`
`    MESSAGE "Simple: " RECID(Simple).`
`    DELETE Simple.`
`    CREATE buf.`
`    MESSAGE "buf:" RECID(buf).`
`    DO TRANSACTION:`
`        MESSAGE "transaction".`
`    END.`
`END.`
	`Simple: 1488`
	`buf: 1483`
	`transaction`
	`Simple: 1490`
	`buf: 1484`
	`transaction`
	`Simple: 1491`
	`buf: 1486`
	`transaction`
	`Simple: 1492`
	`buf: 1488`
	`transaction`
	`Simple: 1493`
	`buf: 1490`
	`transaction`
		the 1488 and 1490 keys **are** reclaimed
		this time the inner DO block is transactional

`DEFINE BUFFER buf FOR Simple. // persistent`
 	
`FOR EACH Simple:`
`    MESSAGE "Simple: " RECID(Simple).`
`    DELETE Simple.`
`    CREATE buf.`
`    MESSAGE "buf:" RECID(buf).`
`    DO:`
`        MESSAGE "transaction".`
`    END.`
`END.`
	`Simple: 1488`
	`buf: 1483`
	`transaction`
	`Simple: 1490`
	`buf: 1484`
	`transaction`
	`Simple: 1491`
	`buf: 1486`
	`transaction`
	`Simple: 1492`
	`buf: 1488`
	`transaction`
	`Simple: 1493`
	`buf: 1490`
	`transaction`
		the 1488 and 1490 keys **are** reclaimed
		this time the inner DO block is **not** transactional

In the last 2 examples, in practice, there are other recids (1491 instead of 1488, etc.). However, I just reused the same output for clarity here - the point is that the recids are still reclaimed.
Also note that this issues appear only for UNDOABLE _temp tables.

While the last 2 examples seem logical to me (the DO / DO TRANSACTION doesn't influence recid reclaiming), for _temp I can't figure out while the TRANSACTION of the inner block is relevant. **Danut's changes are strictly related to _temp, as this is a per-session database and the reclaiming can be implemented without concurrency consideration**. Also, a proper recid reclaiming is required for FOR EACH iterations (if the reclaiming doesn't happen, some FOR-EACH loops may go to infinite). This is also vital for getting FWD-H2 rev. 1.24 back on track.

I am not quite aware for _temp transactions are any different from persistent transactions, so this is why I am looking for some feedback. I also faced a strange test-case that may be insightful here.

Example	Output	Description

`DEFINE TEMP-TABLE tt01 FIELD f1 AS INTEGER INDEX idx1 AS PRIMARY f1.`
`DEFINE BUFFER buf FOR tt01.`
 	
`DO transaction:`
`    CREATE tt01. tt01.f1 = 1.`
`    CREATE tt01. tt01.f1 = 2.`
`END.`
 	
`FOR EACH tt01:`
`    MESSAGE "tt: " RECID(tt01) tt01.f1.`
`    DO TRANSACTION:`
`       DELETE tt01.`
`    END.`
`    CREATE buf.`
`    MESSAGE "buf:" RECID(buf) buf.f1.`
`    UNDO. // this is of interest here`
`END.`
	`tt: 2304`
	`buf: 2304`
	`tt: 2305`
	`buf: 2306`
		the 2304 key is reclaimed in the first iteration, because the delete happens in a (sub?-)transaction
		the first iteration is undo
		the 2305 key is **not** reclaimed, so the new record receives a new recid
		this is a clear example of how FOR-EACH iterations behave differently

The only lead I have right now is that _temp database transactions are dynamic, so 4GL doesn't identify the transaction blocks at compile time. Most probably, FWD has such mechanism (or part of it) implemented, but there are still issues with it, at least for _temp. **I still really feel that recid reclaiming is happening when the delete is actually committed (the transaction ends), but we are still struggling to identify when these _temp transactions actually end**.

#54 - 08/25/2023 10:20 AM - Eric Faulhaber

Alexandru Lungu wrote:

> In the last 2 examples, in practice, there are other recids (1491 instead of 1488, etc.). However, I just reused the same output for clarity here - the point is that the recids are still reclaimed.
> Also note that this issues appear only for UNDOABLE _temp tables.

It is not completely clear to me what your conclusions are regarding persistent tables. I would like to avoid attempting to reclaim primary key values for deleted records in persistent tables, unless you have found it is absolutely necessary for some functionality. The current model uses a unique primary key for every record (across the database, not just per table), and I'd like to keep this simplicity, if possible.

#55 - 08/25/2023 10:46 AM - Alexandru Lungu

- % Done changed from 50 to 80

> It is not completely clear to me what your conclusions are regarding persistent tables. I would like to avoid attempting to reclaim primary key values for deleted records in persistent tables, unless you have found it is absolutely necessary for some functionality. The current model uses a unique primary key for every record (across the database, not just per table), and I'd like to keep this simplicity, if possible.

This is what I intend too for persistent tables. This is not a concern (right now). Persistent tables are shared and the reclaiming itself might cause serious headaches. Hopefully we won't face code that relies on persistent recid reclaiming.

Our concern is for _temp database that is now accessed low-level / using lazy iterations. FWD-H2 from rev. 23 onward causes infinite loops due to the FWD not being able to replicate the proper recid reclaiming. Currently, there are other attempts to fix the infinite loops using more aggressive flushing of _temp records, so this #7535 has a slight change of avoided **for now**.

Danut, please push on 7535c (newly created) a sound solution. Most important is to allow reclaiming when **we absolutely know** that in 4GL the reclaiming happens. This will make the things less buggy:

- Force NO-UNDO eager reclaiming. This is a major step which is also safe.
- Add any changes that are 100% safe - that allows FWD to reclaim only when 4GL does the same.

Also, add to 7535b the patch from #7535-3.

#56 - 08/28/2023 08:05 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, please push on 7535c (newly created) a sound solution. Most important is to allow reclaiming when **we absolutely know** that in 4GL the reclaiming happens. This will make the things less buggy:
>
> - Force NO-UNDO eager reclaiming. This is a major step which is also safe.
> - Add any changes that are 100% safe - that allows FWD to reclaim only when 4GL does the same.
>
> Also, add to 7535b the patch from #7535-3.

Committed 7535c/rev.14710. I've done the following changes:

- Took a few changes from 7535a, mostly changes int he reclaimKeys method.
- Applied the patch from #7535-3 (let me know if you actually meant 7535b).
- Eagerly reclaiming keys when deleting records in a NO-UNDO table.
- Added clearPendingReclaimedKeys which is used by the TemporaryBuffer context to clear pending keys when a transaction rollback is done.

#57 - 08/30/2023 09:39 AM - Alexandru Lungu

- % Done changed from 80 to 100

- Status changed from WIP to Review

Danut, I am mostly OK with the changes in 7535c.

- You call if (!isUndoable()) { reclaimKeys();} Note that this will reclaim all keys across **all** tables. You should reclaim only the pending keys from the no-undo table!! Otherwise, other pending keys from undoable tables will be reclaimed as well.

I am planning to do a set of regression tests and profiling. Please double check that your changes are OK:

- retest a customer application
- run your regression tests
- do some memory checks with VisualVM to ensure that pendingReclaimedKeys and reclaimedKeys are not leaking. **Or you may do a sysout periodically to check if these maps are constantly growing.** Do this check for both undo and no-undo tables.

  Applied the patch from #7535-3 (let me know if you actually meant 7535b).

I meant 7535c. I see it now in 7535c, so it is OK.

I am waiting for your fix and start testing.

#58 - 08/31/2023 09:47 AM - Dănuț Filimon

Alexandru Lungu wrote:

> - You call if (!isUndoable()) { reclaimKeys();} Note that this will reclaim all keys across **all** tables. You should reclaim only the pending keys from the no-undo table!! Otherwise, other pending keys from undoable tables will be reclaimed as well.

Fixed it in **7535c/rev.14711**. I made it possible to choose between which type of tables should the reclaiming be done (all/no-undo/undo) and slightly improved the reclaiming for records that are deleted 1 by 1 using a separate method (only reclaimg recids for a single table at a time, not iterating all the tables anymore).

#59 - 09/04/2023 06:21 AM - Alexandru Lungu

Rebased 7535c with latest trunk and is now at rev. 14725.

#60 - 09/04/2023 08:58 AM - Dănuț Filimon

I tested 7535c using a customer application and there is an issue with direct-access. Now, temp-tables will have a trailing recid component and will cause direct-access to throw an SQLException since the number of columns of the index will not be equal to the number properties that need to match the unique index. This should be an easy fix since we only need to ignore the recid component that was added.

#61 - 09/05/2023 06:48 AM - Dănuț Filimon

> I tested 7535c using a customer application and there is an issue with direct-access. Now, temp-tables will have a trailing recid component and will cause direct-access to throw an SQLException since the number of columns of the index will not be equal to the number properties that need to match the unique index. This should be an easy fix since we only need to ignore the recid component that was added.

This was fixed in **7535a_h2/rev.27**. Note that in #7567-3, Alexandru specified that the primary key column used for direct-access is always named recid, while the name from the directory configuration can be different. Depending on which branch is merged first, the other should be rebased and changed accordingly.

#62 - 09/05/2023 08:24 AM - Dănuț Filimon

I retested a customer application and used VisualVM to keep track of any pendingReclaimedKeys and reclaimedKeys leak but found no evident problems.

#63 - 09/13/2023 04:59 AM - Alexandru Lungu

Merged 7535a_h2/rev.27 into FWD-H2 trunk as rev. 27 and archived.

#64 - 11/06/2023 08:21 PM - Eric Faulhaber

Can this issue be closed?

#65 - 11/07/2023 03:05 AM - Alexandru Lungu

Yes, the changes in 7535c were included already in a separate task branch that was merged to trunk.

#66 - 11/07/2023 03:05 AM - Alexandru Lungu

- Status changed from Review to Test

Files

7535-20230728.patch	3.82 KB	07/28/2023	Dănuț Filimon
infinite-20230822.p	956 Bytes	08/22/2023	Dănuț Filimon