Database - Feature #6630

optimize return of table parameter

07/22/2022 05:55 PM - Ovidiu Maxiniuc

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

Description

History

#2 - 07/22/2022 06:07 PM - Ovidiu Maxiniuc

While working on a related issue I noticed the following stack trace:

```
        at org.h2.jdbc.JdbcConnection.prepareStatement(JdbcConnection.java:734)
        at com.goldencode.p2j.persist.orm.TempTableDataSourceProvider$DataSourceImpl$1.prepareStatement(TempTa
bleDataSourceProvider.java:226)
        at com.goldencode.p2j.persist.orm.SQLQuery.scroll(SQLQuery.java:279)
        at com.goldencode.p2j.persist.orm.Query.scroll(Query.java:231)
        at com.goldencode.p2j.persist.Persistence.scroll(Persistence.java:1280)
        at com.goldencode.p2j.persist.PreselectQuery.executeScroll(PreselectQuery.java:5546)
        at com.goldencode.p2j.persist.PreselectQuery.executeQuery(PreselectQuery.java:5520)
        at com.goldencode.p2j.persist.AdaptiveQuery.executeQuery(AdaptiveQuery.java:3472)
        at com.goldencode.p2j.persist.PreselectQuery.execute(PreselectQuery.java:5176)
        at com.goldencode.p2j.persist.AdaptiveQuery.execute(AdaptiveQuery.java:3359)
        at com.goldencode.p2j.persist.AdaptiveQuery.stateChanged(AdaptiveQuery.java:2896)
        at com.goldencode.p2j.persist.ChangeBroker.stateChanged(ChangeBroker.java:645)
        at com.goldencode.p2j.persist.RecordBuffer.reportChange(RecordBuffer.java:8587)
        at com.goldencode.p2j.persist.RecordBuffer.delete(RecordBuffer.java:7902)
        at com.goldencode.p2j.persist.TemporaryBuffer.delete(TemporaryBuffer.java:6773)
        at com.goldencode.p2j.persist.RecordBuffer.delete(RecordBuffer.java:7711)
        at com.goldencode.p2j.persist.TemporaryBuffer.lambda$loopDelete$15(TemporaryBuffer.java:6482)
        at com.goldencode.p2j.util.Block.body(Block.java:636)
        at com.goldencode.p2j.util.BlockManager.processForBody(BlockManager.java:8662)
        at com.goldencode.p2j.util.BlockManager.forEachWorker(BlockManager.java:10424)
        at com.goldencode.p2j.util.BlockManager.forEach(BlockManager.java:4357)
        at com.goldencode.p2j.persist.TemporaryBuffer.loopDelete(TemporaryBuffer.java:6473)
        at com.goldencode.p2j.persist.TemporaryBuffer.copyAllRows(TemporaryBuffer.java:3482)
        at com.goldencode.p2j.persist.TemporaryBuffer.copyAllRows(TemporaryBuffer.java:3277)
        at com.goldencode.p2j.persist.TemporaryBuffer.performOutputCopy(TemporaryBuffer.java:6215)
        at com.goldencode.p2j.persist.OutputTableCopier.finished(OutputTableCopier.java:102)
        at com.goldencode.p2j.util.ProcedureManager$WorkArea.scopeFinished(ProcedureManager.java:5020)
        at com.goldencode.p2j.util.TransactionManager.processScopeNotifications(TransactionManager.java:7432)
        at com.goldencode.p2j.util.TransactionManager.popScope(TransactionManager.java:4607)
        at com.goldencode.p2j.util.TransactionManager.access$7000(TransactionManager.java:654)
        at com.goldencode.p2j.util.TransactionManager$TransactionHelper.popScope(TransactionManager.java:8536)
        at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:8310)
        at com.goldencode.p2j.util.BlockManager.internalProcedure(BlockManager.java:611)
        at com.goldencode.p2j.util.BlockManager.internalProcedure(BlockManager.java:597)
        at <customer code: end of block>
```

The customer code calls a internal procedure which has an input-output table parameter. You can see that the procedure block end and the OutputTableCopier is doing its job of populating the table parameter. Because it is not in append mode, the old content is deleted, in a loop. For each deleted record the state listeners are invoked and, in this case, the AdaptiveQuery.stateChanged() does the following (around line 2888): invalidates components, results and invokes execute() to repopulate the preselected result. In a worst case scenario, this happens for each deleted record in the table parameter. Which means a lot of SQL queries which are executed but the result is immediately dropped.

I am aware that this is a preselect query and the backing result set must be there, but I wonder whether we could act lazy in this case: to delay the

effective re-fetch of the query only after the delete operation is finished on the table parameter. Or even better, to delay the execute() up to when the next row is requested. Of course, its buffers might be in an undefined state, but this should only happen during the 'internal' house-keeping and they should bot be exposed to ABL converted code.

More details in #6061-75.

#3 - 04/27/2023 03:47 AM - Alexandru Lungu

- Assignee set to Dănuț Filimon

#4 - 04/27/2023 03:50 AM - Alexandru Lungu

Danut, I recall I changed the AdaptiveQuery from TemporaryBuffer.loopDelete into a PreselectQuery. Basically, there is no invalidation happening anymore.
Please try to recreate the scenario in #6630 (maybe based on #6061-75) and check if this is optimal. If PreselectQuery is acceptable there performance-wise, we can close this.

#5 - 05/03/2023 09:51 AM - Dănuț Filimon

I recreated a similar scenario where TemporaryBuffer.loopDelete is called using a class field that has a destructor. There is no invalidation happening anymore as it was specified and I found no problems while testing, so the change fits this case correctly. This task can be closed.

#6 - 05/30/2023 05:17 AM - Alexandru Lungu

- Status changed from New to WIP

- % Done changed from 0 to 30

This makes sense to me. TemporaryBuffer.loopDelete can't ever interact with other business logic that trigger record changes, thus invalidate. Triggers aren't available for temporary tables and there shouldn't be any side effect that alters the currently iterated table. Therefore, it doesn't make sense (functionally) to use AdaptiveQuery for this task; PreselectQuery is good enough for this task.

Even so, we have recent changes to AdaptiveQuery on single-table that use lazy mode, disallowing invalidation. In fact, #7061 shown that lazy is slightly faster than preselect (due to a lower memory foot-print) in simple cases.

Before closing this, we should do a comparison between PreselectQuery and AdaptiveQuery on this scenario. Please make a dummy 4GL test with a single temp table (with multiple indexes and several fields). Populate and run empty-temp-table lots of time. Track only empty-temp-table. If AdativeQuery is faster, we will go back to the old implementation now that we have #7061.

#7 - 05/31/2023 07:01 AM - Dănuț Filimon

I redesigned my test case and profiled both AdaptiveQuery and PreselectQuery. I used a temp-table with 100 fields (50 integer and 50 class objects) and 10 indexes distributed 50/50 on both integers and class objects. The tested the following cases:

- 100 records, 5 and 10 iterations
- 1000 records, 5 and 10 iterations
- 2500 records, 5 and 10 iterations
- 5000 records, 5 and 10 iterations
- 10000 records, 5 and 10 iterations

Based on the average results of the iterations, I obtained the the following results for EMPTY TEMP-TABLE operation:

PreselectQuery	5 iterations (ms)	10 iterations (ms)
100	89.4	36

records		
1000 records	280	256.7
2500 records	652.6	669.1
5000 records	1469.4	1466.7
10000 records	3033.4	2938.4

AdaptiveQuery	5 iterations (ms)	10 iterations (ms)
100 records	89	38.6
1000 records	284.6	246.9
2500 records	620.2	622.4
5000 records	1366.6	1508.1
10000 records	2778.2	2740.6

And the difference between PreselectQuery and AdaptiveQuery is:

Difference	5 iterations (%)	10 iterations (%)
100 records	-0.447%	+7.222%
1000 records	+1.642%	-3.817%
2500 records	-4.964%	-6.979%
5000 records	-6.996%	+2.822%
10000 records	-8.413%	-6.731%

From what I've tested, AdaptiveQuery doesn't invalidate at all. AdaptiveQuery.invalidateIfReferenceRowExists is called, but there is no cursor available, other invalidation methods are not called at all. The results are overall better for AdaptiveQuery, what do you think Alexandru?

#8 - 06/01/2023 04:45 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 30 to 100

Usually, temp-tables won't hold 10.000 records from what I have seen. We should rely on the 100/1000 tests with 5/10 iterations. At that point, I think they are quite close to each other with a slight preference for PreselectQuery, so I will leave it as now.

cursor is not available as the query is non-scrolling, which is the right configuration.

#9 - 06/23/2023 03:12 AM - Alexandru Lungu

- Status changed from Review to Test

This can be closed now. #6630-2 is already solved and merged into trunk at some point.