Database - Bug #3220

(TEMP-TABLE) rows get iterated twice in certain conditions

12/12/2016 01:49 PM - Ovidiu Maxiniuc

Status:	Closed	**Start date:**	12/12/2016
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#2 - 12/12/2016 02:00 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

I have found in client code code similar to this one:

```
DEFINE TEMP-TABLE ttx
   FIELD s AS INTEGER
   FIELD n AS INTEGER
   FIELD c AS CHAR.

CREATE ttx. n = 1. c = "111".
CREATE ttx. n = 2. c = "222".
CREATE ttx. n = 3. c = "333".
CREATE ttx. n = 4. c = "444".
CREATE ttx. n = 5. c = "555".
CREATE ttx. n = 6. c = "666".
CREATE ttx. n = 7. c = "777".
CREATE ttx. n = 8. c = "888".

DEFINE VAR l AS INT.

l = 0.
FOR EACH ttx WITH 16 DOWN NO-LABELS:
   DISPLAY "Processing" ttx.s ttx.n ttx.c.
   l = l + 1.
   IF l = 5 THEN
      ASSIGN n = 0
             c = "000".
   ELSE IF l > 5 THEN DO:
      DELETE ttx.
      DISPLAY "del".
   END.
END.
```

This should filter out the last 3 rows, altering the 5th, in a single iteration. I cannot say at this moment the exact conditions but, FWD is looping again the previously iterated rows, so the output is this:

```
┌──────────────────────────────────────────┐
|Processing          0          1 111      |
|Processing          0          2 222      |
|Processing          0          3 333      |
|Processing          0          4 444      |
```

```
|Processing          0          5 555          |
|Processing          0          6 666      del|
|Processing          0          7 777      del|
|Processing          0          8 888      del|
|Processing          0          1 111      del|
|Processing          0          2 222      del|
|Processing          0          3 333      del|
|Processing          0          4 444      del|
|Processing          0          0 000      del|
|                                              |
|                                              |
|                                              |
└──────────────────────────────────────────────┘
```

In consequence, at the end of the procedure the temp-table remains empty, instead of the expected first 5 records.

The content of first IF is not important. However, the exact figured of 5 and 8 seems to be. I wonder if this is perhaps related to the incremental batch size (in Fibonacci increments) for row retrieval has something to do with this issue.
I did not test the permanent tables yet. Working on this.

#3 - 12/12/2016 02:09 PM - Eric Faulhaber

What is the Progress output for the same test?

#4 - 12/12/2016 02:13 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> What is the Progress output for the same test?

Progress outputs as expected:

```
┌──────────────────────────────────────────────┐
|Processing          0          1 111          |
|Processing          0          2 222          |
|Processing          0          3 333          |
|Processing          0          4 444          |
|Processing          0          5 555          |
|Processing          0          6 666      del|
|Processing          0          7 777      del|
|Processing          0          8 888      del|
|                                              |
|                                              |
|                                              |
|                                              |
|                                              |
|                                              |
|                                              |
|                                              |
└──────────────────────────────────────────────┘
```

#5 - 12/12/2016 02:36 PM - Eric Faulhaber

I believe the delete switches the adaptive query from preselect into dynamic mode. Sorting uses the primary key in the absence of an explicit index. These should be sequential and ascending in the order the records are created in this test case. I don't know why this is resetting to visit the first record again after the last is deleted.

#6 - 12/12/2016 03:28 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> I believe the delete switches the adaptive query from preselect into dynamic mode. Sorting uses the primary key in the absence of an explicit index. These should be sequential and ascending in the order the records are created in this test case. I don't know why this is resetting to visit the first record again after the last is deleted.

Indeed, when deleteRecord is called, the AdaptiveQuery adapter's cursor is invalidated but position not preserved. Here is the call stack:

```
    at com.goldencode.p2j.persist.AdaptiveQuery.invalidate(AdaptiveQuery.java:3609)
    at com.goldencode.p2j.persist.AdaptiveQuery.invalidate(AdaptiveQuery.java:3799)
    at com.goldencode.p2j.persist.AdaptiveQuery.stateChanged(AdaptiveQuery.java:2656)
    at com.goldencode.p2j.persist.ChangeBroker.stateChanged(ChangeBroker.java:593)
    at com.goldencode.p2j.persist.RecordBuffer.reportChange(RecordBuffer.java:6977)
    at com.goldencode.p2j.persist.RecordBuffer.reportChange(RecordBuffer.java:6819)
    at com.goldencode.p2j.persist.RecordBuffer.delete(RecordBuffer.java:6314)
    at com.goldencode.p2j.persist.TemporaryBuffer.delete(TemporaryBuffer.java:3588)
    at com.goldencode.p2j.persist.BufferImpl.deleteRecord(BufferImpl.java:3162)
    at com.goldencode.p2j.persist.BufferImpl.deleteRecord(BufferImpl.java:594)
```

Then, when the next is called, it will create a new DynamicQuery (line 3168) from the invalidComponent. The first two of the new query uses and ttx1_1impl0_.id>? predicate to correctly advance twice, but the 3rd time it will fail to add it so the whole table is iterated again.

#7 - 12/13/2016 03:13 PM - Ovidiu Maxiniuc

After further debugging. Apparently the AdaptiveQuery is quite eager to return back to progressive/preselect mode. This is normal, I guess, because it offers the maximum performance. The only issue I noticed is that the management of the breakValue. At a second conversion from random access to preselect mode, the breakValue is lost. In consequence, the query will not be constructed with the constraint to skip the already processed rows, so the whole table is refetched from database, in this particular case, making all of them to get deleted.

#8 - 12/14/2016 03:22 PM - Ovidiu Maxiniuc

Eric,
I have some difficulties to find the exact cause why the breakValue is deleted.
In the event of deleting the record currently active in query, we are switching to dynamic mode. Then later we try to improve performance and attempt to go progressive. This seems fine.
I have some suspicions about the finally block from AdaptiveQuery.execute() (around line 3150). Why are we dropping the break value at after switching back to preselect mode? In my understanding, the breakValue should be kept and even might need to be updated to reflect the last processed row, so it and previous ones to be skipped in the new progressive result. If the current query was never dynamic, then the breakValue shouldn't have been initialized. The fact that the line is there means it was useful in some cases. The line is not documented so I don't know which one it is in order to reset the value only for those cases.

#9 - 01/12/2017 07:44 AM - Ovidiu Maxiniuc

During extensive testing, the patch held, no regressions noticed.
The change is part of 11135 revision, committed with parent task (branch 3201c). This issue can be closed.

#10 - 01/12/2017 07:46 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed

- Assignee set to Ovidiu Maxiniuc

- Target version set to Cleanup and Stablization for Server Features