Database - Bug #11873

Dynamic single-table results become scrolling after invalidation

09/15/2026 03:23 AM - Alexandru Lungu

Status:	Internal Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Alexandru Lungu	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	Ovidiu Maxiniuc
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

History

#3 - 09/15/2026 04:00 AM - Alexandru Lungu

- % Done changed from 0 to 100

- Status changed from New to Review

- Assignee set to Alexandru Lungu

- reviewer Ovidiu Maxiniuc added

There is a bug from #9724 changes.

Before #9724, all queries used as dynamic results for AdaptiveQuery were inheriting the scrolling trait.

After #9724 (indexed-reposition implementation), RAQ is no longer scrolling if used as a results provider for a AQ. This was because the cursor of the parent query and the cursor of the child query were conflicting after indexed repositioning. Also, the RAQ in this case is used only as a delegate to retrieve data using the FQL bundle, not to actually cache results - the caching is already done by the AQ. In 9724c/rev. 15856, the fix landed because in indexed-reposition, a first on the RAQ would deliver the first in the cursor instead of the first relative to the indexed-reposition forced by the AQ.

The decision: the AQ is always driving the caching in cursor and the queries used to provide results are always providing non-cached results.

There is however a leftover:

- DynamicQuery.resetScrolling is making the cursor empty **even if the query was not scrolling in the first place**. It is called by AQ.resetDynamicScrolling
  - called by stateChanged when invalidating. In this case, if the query was dynamic and it was meant to be invalidated, then its cursor was reset.
  - called by invalidate when a forceful invalidate occurs. In this case, if the query was dynamic and it was meant to be invalidated, then its cursor was reset.

In both cases above, the cursor **shouldn't be reset** if the underlying query **was not scrolling**. The side-effect was that non-scrolling RAQ **became** scrolling on invalidation. I detected this when using a AQ.next which snapshot the scrolling attribute (false), running the query (which made the buffer flush, stateChanged being called and the delegate becoming scrolling) and setting the off-end. The off-end was consulted based on the scrolling flag (which was snapshot as false), but the real off-end trait was now delivered by the pristine cursor. This made a query without records report that records were found. In a 130 unit test-suite with 6 failures, a fix for this made all 130 tests pass.

PS: setScrolling was calling resetScrolling. I made setScrolling to set the cursor directly if the query is not scrolling and resetScrolling to reset the cursor if the query **is already** scrolling.

Ovidiu, please review. Committed 11873a/16755.

#4 - 09/15/2026 07:53 AM - Ovidiu Maxiniuc

- Status changed from Review to Internal Test

Nice catch!

The changes from branch make perfect sense. I would keep the resetScrolling() method back into the protected methods section of the class.