Database - Bug #7797

FWD-H2 LAZY reverse index regression

09/13/2023 10:37 AM - Alexandru Lungu

Status:	Closed	**Start date:**	
Priority:	High	**Due date:**	
Assignee:	Radu Apetrii	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 09/13/2023 10:45 AM - Alexandru Lungu

```
=== modified file 'src/main/org/h2/command/dml/Select.java'
--- src/main/org/h2/command/dml/Select.java    2023-05-24 07:08:23 +0000
+++ src/main/org/h2/command/dml/Select.java    2023-06-30 10:44:58 +0000
@@ -873,7 +873,7 @@
         }
         // Do not add rows before OFFSET to result if possible
         boolean quickOffset = !fetchPercent;
-        if (sort != null && (!sortUsingIndex || isAnyDistinct())) {
+        if (sort != null && !lazy && (!sortUsingIndex || isAnyDistinct())) {
             result = createLocalResult(result);
             result.setSortOrder(sort);
             if (!sortUsingIndex) {
```

This (trunk/rev. 24) is causing a regression.

The issue is that a reverse index is used in the sort clause (ex: multiplex desc, f1 desc), even though there is an index on @f1 asc. In this case, sortUsingIndex is false and such the FWD-H2 will return the records in an arbitrary order. For correctness, H2 was doing a final sort. However, when using lazy, we don't have the luxury of a final sort, so we **must** return the records in the correct order right from the beginning.

This is usually true, as we always emit lazy only for AdaptiveQuery, which works exclusively on tree indexes. **However**, we are missing the scenario where a reverse index is used. Note the following code from Select.queryWithoutCache:

```
result.setReversed(!sortUsingIndex && reversedIndex == topTableFilter.getIndex());
```

We are not doing setReversed for lazy queries, just for normal queries.

Radu, please make sure your LazyResultQueryFlat is actually returning the records in reversed order if !sortUsingIndex && reversedIndex == topTableFilter.getIndex(). This can be done by routing the previous calls into next, next into previous, last into first and first into last. Maybe you can even do a result wrapped, something like ReversedResults class that proxies LazyResultQueryFlat and does the routing.

Please note that this is a regression in a customer application and should be fixed asap.

#2 - 09/14/2023 03:10 AM - Radu Apetrii

I'm on it. I think I already have the changes necessary, but they need to be added to 7066c_h2 (which I will create very soon).

Alex, by chance, do you have an example where the records are retrieved in arbitrary order?

#3 - 09/14/2023 03:30 AM - Alexandru Lungu

I don't have an example yet.

Attempt to make something like FOR EACH tt BY tt.f1 DESC, while you have an ASC index on f1. Populate the table with ~10 elements of different f1. Most probably, FWD will return them in a random order.

#4 - 09/14/2023 05:19 AM - Radu Apetrii

This might turn out to be more messy than I hoped. I was right saying that the changes are there, but it's a bit more complicated to extract them. The logic for reversed indexes is strictly connected to use index, the keyword we would enforce in order to let H2 know that it needs to use a specific index. At the moment, I see two options to handle this:

- **Integrate the changes from 7066a and 7066a_h2.** Note that they are dependent on each other, so just the changes from 7066a_h2 won't do the trick.
  - Advantages:
    - The logic is already implemented.
    - We could allow the use of use index for just single-table queries until we test further.
    - We deal with #7066 once and for all.
  - Disadvantages:
    - Quote: My point here is that these 2 tasks should be merged together. (#7066 and #6582). We would make an exception to this statement and (maybe) integrate just #7066 for now.
- **Implement a ReversedResults class** that deals with this for now.
  - Advantages:
    - We would solve this issue without great effort.
    - The FWD project won't suffer changes (I think).
    - We would integrate #7066 and #6582 at the same time, later.
  - Disadvantages:
    - It will (most probably) be harder to integrate the changes from #7066 after.
    - A big part of the logic that is already implemented will probably go to waste.

Personally, I lean more towards the first solution, but I will wait for your opinion too. If you choose the same path, I will write some details about the integration process of 7066a_h2.

#5 - 09/14/2023 05:36 AM - Alexandru Lungu

Radu Apetrii wrote:

> This might turn out to be more messy than I hoped. I was right saying that the changes are there, but it's a bit more complicated to extract them. The logic for reversed indexes is strictly connected to use index, the keyword we would enforce in order to let H2 know that it needs to use a

specific index. At the moment, I see two options to handle this:

- **Integrate the changes from 7066a and 7066a_h2.** Note that they are dependent on each other, so just the changes from 7066a_h2 won't do the trick.
  - Advantages:
    - The logic is already implemented.
    - We could allow the use of use index for just single-table queries until we test further.
    - We deal with #7066 once and for all.
  - Disadvantages:
    - Quote: My point here is that these 2 tasks should be merged together. (#7066 and #6582). We would make an exception to this statement and (maybe) integrate just #7066 for now.

~~Finish #7066 is way out of reach. #7066 is about converting to AdaptiveQuery instead of CompoundQuery **especially** for persistent tables. Having to deal with the temporary database is a side job there.~~ I was thinking of #6582, sorry.
I agree that use-index is something that could be helpful here, but it is a large piece of codes requiring testing. **Currently, there are some regressions to be fixed, so we don't have the luxury to introduce even more functionality.**

- **Implement a ReversedResults class** that deals with this for now.
  - Advantages:
    - We would solve this issue without great effort.
    - The FWD project won't suffer changes (I think).
    - We would integrate #7066 and #6582 at the same time, later.
  - Disadvantages:
    - It will (most probably) be harder to integrate the changes from #7066 after.
    - A big part of the logic that is already implemented will probably go to waste.

You can think as a temporary solution if it requires minimal effort. Also, having implementation going to waste is not a problem if we can integrate them and have the best of both worlds in the end. In other words, if ReversedResults provides us clean and consistent results, we should **actually** consider it instead of 7066a_h2.
Also, meeting the tomorrow deadline of integrating FWD-H2 is not quite feasible with a large and untested set of changes.

> Personally, I lean more towards the first solution, but I will wait for your opinion too. If you choose the same path, I will write some details about the integration process of 7066a_h2.

Can you elaborate what is the code waste in the second approach? I expected to have something similar to ReversedResults in 7066a_h2 already. What was the alternative implementation substituting ReversedResults? Can you do a theoretical performance and code maintenance comparison between the 7066a_h2 approach and a potential ReversedResults?

> I've added support for queries that use both lazy and use_index. Because of this, some logic in LazyResult had to change in order to match the concept of reversed iteration of records. It might seem weird, but some next() calls actually mean prev() and vice versa.

This is something mentioned in #7066-16. This is the part that would have been solved by a ReversedResults. What was used there?

#6 - 09/15/2023 06:30 AM - Radu Apetrii

I put some thought into this and tried a couple of things and I think (or hope at least) that I've reached a pretty "amiable" solution, meaning that when the changes from #7066 will be merged, we won't have too much trouble integrating them. **The commit is on 7066c_h2, rev. 29.**
I've added a ReversedResults, which is a class that contains a LazyResult, but it iterates the records in reversed order (prev is actually next and vice versa). I've tested an example with by tt.f1 desc, as you mentioned, and the result was OK. Also, I added some tests with this exact case.

Alex, can you please take a look at the changes and see if the regression from the customer application is fixed?

#7 - 09/15/2023 10:37 AM - Alexandru Lungu

- % Done changed from 0 to 80

Review:

- You made sortUsingReversedIndex by default true. I don't think this is right!
- For ReversedResults: isAfterLast should call isBeforeFirst instead. Alos, isBeforeFirst is not extended. If it is not implemented yet, consider adding it to ResultInterface. If there are lot of implementations, you can add a default that throws an illegal state exception.
- Maybe you should throw illegal state exceptions if you call the default implementation.

I am OK with the changes overall. Maybe is it quite confusing the state in which we actually are when optimizing the index choosing:

- if we find an exact index, we set it and use sortUsingIndex
- if we find an exact reversed index, we forcefully set sortUsingReversedIndex and eventually also set sortUsingIndex and the index itself to the table.
- if we don't find any of these two, I think everything is alright.

My point is that sortUsingIndex and sortUsingReversedIndex are not quite uniformly handled. I am looking forward to actually change getIndex to return something like a pair: the index and if it is used for reversed sorting or not. This way, you should double check the usages of sortUsingIndex to augment them with sortUsingReversedIndex and sort accordingly. sortUsingIndex || !sortUsingReversedIndex is not a semantically logic clause.

#8 - 09/21/2023 03:02 AM - Radu Apetrii

This turned out to be quite a massive set of changes. While doing some tests I noticed things that were not in order with ReversedResults. Thus, I added to **7066c_h2, rev. 30** support and tests for OFFSET and LIMIT and fixed some issues that appeared with the rowId's value. The changes include:

- The rowId in ReversedResults should point to after the last record and not before the first one, as in LazyResults.
- Took care of LIMIT cases, because ReversedResults would find the next record, even though the limit would be reached.
- Added IndexDelegate which handles the sortUsingIndex and sortUsingReversedIndex cases, with their respective indexes.
- Allowed the creation of ReversedResults only when we have lazy.

#9 - 09/25/2023 10:18 AM - Alexandru Lungu

Review of 7066c_h2:

- I like seeing ~20 new tests on LAZY :)
- Why is currentSearchRow = null required in your changes?
- I think createShallowCopy should also return a ReversedResults instance. Thus, return new ReversedResults(result.createShallowCopy(targetSession).
- ascendingOrder is not quite accurate for IndexDelegate. Rename it to reversedOrder for the opposite semantic. The index can be either asc. or desc. while the order by can be either asc. or desc.
- Boolean getters are named using is as a prefix. Thus, rename getUsingReversedIndex into isUsingReversedIndex. The same goes for getUsingIndex and getUsingAscendingIndex. Maybe rename AscendingIndex to OrderedIndex (note that the index can be defined with desc, but still be the "ascending index").
- int sortType = sort != null ? sort.getSortTypes()[0] : 0; can you clarify this addition? Is the sortType 0 for asc and 1 for desc? In this case, note that you can still use the ordered index (or as you name it "ascending index") if its components match the direction which can be either asc or desc.

Overall: reversed index usage means the index that has the same fields as the order by, but with exact opposite direction (e.g. asc in index, desc in order by **or** desc in index, asc in order by).

#10 - 09/26/2023 04:45 AM - Radu Apetrii

I've addressed the review from the previous post and committed the changes to 7066c_h2, rev. 31.

> - Why is currentSearchRow = null required in your changes?

It is not required. When I started implementing the solution, this was part of the logic, but it became obsolete in the meantime. I deleted those two lines.

> - int sortType = sort != null ? sort.getSortTypes()[0] : 0; can you clarify this addition? Is the sortType 0 for asc and 1 for desc? In this case, note that you can still use the ordered index (or as you name it "ascending index") if its components match the direction which can be either asc or desc.

This addition is related to quickOffset. I wanted to allow the use of quickOffset only in the case of index asc, order by asc. For the other cases, the intention is to do the offset after the records are "sorted". I also tried to allow quickOffset = true for index desc, order by desc, but there was some strange behavior, so I gave up on the idea.

I re-tested with testFWD which includes TestLazy and there were no errors.

#11 - 09/27/2023 05:43 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 80 to 100

I am OK with the changes in 7066c_h2.
I also changed the planner a bit to score more the indexes that are reversed in direction to honor your changes. This way, the H2 planner will eventually prefer to pick reversed indexes (if any), as it is now aware that they can be solved faster.

The regression is gone in my tests and everything is back to normal.
Merged 7066c_h2 to trunk as rev. 30.

#12 - 09/27/2023 05:44 AM - Alexandru Lungu

- Status changed from Review to Test

#13 - 09/27/2023 08:03 AM - Greg Shah

Is this ready to be deployed on proj.goldencode.com? If so, is it safe to push it with the current version of trunk?

#14 - 09/27/2023 09:04 AM - Alexandru Lungu

I would like to have 7789a_h2 merged as well, before the upgrade. I finished the implementation (going to update #7789 now), but I still need to do the testing.
Thus, I am planning the rev. 31 commit as the merge of 7789a_h2. At that point, I would go ahead with some extensive testing and provide you a green-light to do the upgrade.

With rev. 30 things are quite good with my testing.

#15 - 09/27/2023 10:12 AM - Greg Shah

Sounds good. Eric will handle the push when you are ready.

#16 - 10/30/2023 06:34 AM - Alexandru Lungu

This reached trunk in fwd-h2-1.33-trunk and can be closed.

#17 - 10/30/2023 07:22 AM - Greg Shah

- Status changed from Test to Closed