Database - Bug #4945

Optimize indexes for H2 dialect

10/06/2020 10:48 AM - Ovidiu Maxiniuc

Status:	Feedback	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#1 - 10/06/2020 10:54 AM - Ovidiu Maxiniuc

We notice that sometimes the temp-table queries are slow because the matching reversed index is not selected by the query plan. In consequence, the result set is re-sorted in H2 driver before returning to caller.

This task was created to collect a survey of the H2 code (1.4.200, as modified by GCD) which manages fetching and sorting results using an index and possible discussion about a fast implementation which eventually might be accepteed back into H2 main code base.

#2 - 10/06/2020 11:18 AM - Ovidiu Maxiniuc

I took a look over H2 code that handles selection of the index (Select.prepare())to be used for a query and the post-processing (LocalResultImpl.done()). Here are a couple of initial observations:

- the Index that drives the query is computed in Select.getSortIndex(). The method takes the list of sort expressions and attempt a match to each known index. To note that the nulls position is taken into consideration for each index component. So we must match exactly. Also, there is no reversed index check. I think a check can easily added by using a SortOrder matrix. If the first component is a match but the direction is reversed we set a flag and check the rest of components for reversed order. IIRC, we drop a legacy reversed index so when selecting a reversed one we are sure there is no other match. It appears that the offset and limit are applied to the LocalResultImpl just before returning to caller. Based on the previous flag (reversed index) we can do the appropriate result set windowing when sorting or applying offset and limit.
- I discovered a bug in FWD related to sorting temp-tables queries. When a reversed order is detected, the nulls first is correctly used for ascending components. However, The structure of temp-table database indexes also contains the _multiplex as the first index component. It is never reversed so temp-table driver will not be able to correctly detect the reversed index.

#3 - 10/06/2020 12:13 PM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> - I discovered a bug in FWD related to sorting temp-tables queries. When a reversed order is detected, the nulls first is correctly used for ascending components. However, The structure of temp-table database indexes also contains the _multiplex as the first index component. It is never reversed so temp-table driver will not be able to correctly detect the reversed index.

I don't understand - what's the exact ORDER BY clause? Do you mean there is an ORDER BY _multiplex, tt1.f1 DESC NULLS FIRST ... and the ORDER BY doesn't add DESC to the _multiplex?

If so, I don't see a reason why not adding DESC as needed. We add the _multiplex to the ORDER BY because the WHERE will include a _multiplex = ?, and otherwise (for non-reversed cases), H2 will not be able to match the index to the ORDER BY, if _multiplex is not used.

#4 - 10/06/2020 12:24 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> I don't understand - what's the exact ORDER BY clause? Do you mean there is an ORDER BY _multiplex, tt1.f1 DESC NULLS FIRST ... and the ORDER BY doesn't add DESC to the _multiplex?

Exactly. The index is created as:

```
create index idx__tt23_ix1__${1} on tt23 (_multiplex, f1 nulls last) transactional;
```

The query is sorted by:

```
[...]
order by
    tt1_3_1__i0_._multiplex asc, tt1_3_1__i0_.f1 desc nulls first
```

Clearly it will not be a match.

> If so, I don't see a reason why not adding DESC as needed. We add the _multiplex to the ORDER BY because the WHERE will include a _multiplex = ?, and otherwise (for non-reversed cases), H2 will not be able to match the index to the ORDER BY, if _multiplex is not used.

Right. I am adding now the DESC direction to _multiplex when FWD detects a reversed index.

#5 - 10/06/2020 12:25 PM - Constantin Asofiei

What happens if the index in 4GL is originally as index ix1 f1 desc?

#6 - 10/06/2020 12:27 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Right. I am adding now the DESC direction to _multiplex when FWD detects a reversed index.

But will this mean H2 will select the index AND not re-sort after the results are fetched? Or just that H2 will use the index to retrieve the records, but still require a re-sort in the reverse direction after the results are retrieved?

#7 - 10/06/2020 01:07 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> What happens if the index in 4GL is originally as index ix1 f1 desc?

Evidently, the index DDL declaration changes to:

```
create index idx__tt23_ix1__${1} on tt23 (_multiplex, f1 desc nulls first) transactional;
```

But the query stays the same: order by tt1_3_1__i0_._multiplex asc, tt1_3_1__i0_.f1 desc nulls first. In this case we have a match.

#8 - 10/06/2020 01:11 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> But will this mean H2 will select the index AND not re-sort after the results are fetched? Or just that H2 will use the index to retrieve the records, but still require a re-sort in the reverse direction after the results are retrieved?

If it woks as I expect, H2 will return the result reverse-sorted. When the result is "done" we just need to reverse its rows. This is a far faster operation than a full sort based on given criteria. And eventually "trim" the result to windowing/paging parameters specified by limit and offset options.

#9 - 10/06/2020 01:20 PM - Ovidiu Maxiniuc

Another issue I noticed while debugging: there was an extra index, having the recid as unique component. It was marked as the primary index. The truth is, none of the other indexes were explicitly designed as primary in ABL code. But, IIRC, the first defined index should be automatically promoted to primary.

The existence of this might not be a problem, but it simply adds a new index to management which will increase the times for INSERT operations. It is only used when querying the table in find-by-rowid mode. It is difficult at this moment for me to decide which is more advantageous.

#10 - 10/06/2020 03:36 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Constantin Asofiei wrote:
>
> > What happens if the index in 4GL is originally as index ix1 f1 desc?
>
> Evidently, the index DDL declaration changes to:
> [...]
>
> But the query stays the same: order by tt1_3_1__i0_._multiplex asc, tt1_3_1__i0_.f1 desc nulls first. In this case we have a match.

Since _multiplex has no business meaning and should have the same value across all rows of a result set on any converted temp-table query, it doesn't matter which way we sort _multiplex, so long as it is consistent between the ORDER BY clause and the way the index is created in H2. Are you saying there are cases where there is a mismatch currently, or do we match between the ORDER BY and the index definition in all cases?

When we tack recid onto the end of an index and an ORDER BY clause, the sort direction **does** matter. The direction of this component should invert if the rest of the index components are inverted. Again, it is critical that the ORDER BY and the index definition are consistent. Do you know of any cases where they are not?

These statements are not news to anyone, but I want to be sure we correct any cases where we do not follow these rules today, if we have not already.

#11 - 10/06/2020 04:17 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Eric Faulhaber wrote:
>
> > But will this mean H2 will select the index AND not re-sort after the results are fetched? Or just that H2 will use the index to retrieve the records, but still require a re-sort in the reverse direction after the results are retrieved?
>
> If it woks as I expect, H2 will return the result reverse-sorted. When the result is "done" we just need to reverse its rows. This is a far faster operation than a full sort based on given criteria. And eventually "trim" the result to windowing/paging parameters specified by limit and offset options.

Are you talking about H2 changes here, or just FWD changes?

If H2, how much do you think the "we just need to reverse its rows" actually costs? Nothing is free.

Have you looked at the code which actually retrieves the rows? Do you think it is feasible (i.e., with reasonable effort/safety) to modify this code so that it actually retrieves the rows in the reversed order already, using the "inverted" index?

How does limiting work in H2? If it uses an index to retrieve the records, is it smart enough to apply the limit before retrieving all the rows? Or does it retrieve all the rows first, then apply the limit? I understand that in the case of an ORDER BY not matching the retrieval index, it will have to retrieve all the rows and sort before applying the limit, but I am asking about the optimized case, where the retrieval index already orders the records the same as the ORDER BY.

#12 - 10/06/2020 05:00 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Are you talking about H2 changes here, or just FWD changes?

There are two set of changes:

- one in FWD: as you see in note-4 above, the query order by tt1_3_1__i0_._multiplex asc, tt1_3_1__i0_.f1 desc nulls first will not match the (_multiplex, f1 nulls last) index. This example is taken from Constantin testcase #4785-962 - 1st commented as "this will re-sort the rows". If we want to have a match on reversed index the _multiplex sort component must be injected with the direction of the index detected from the other components.
- the second in H2: I do not see any attempt in org.h2.command.dml.Select.getSortIndex() to detect reversed indexes. We need to add this, or the results will not be sorted server-side. H2 will probably use some index and provide the result in that order. This is a bit of puzzle for me. We are doing our best to match the inversed index, but H2 ignores it if it cannot match a direct index and does a simple sort on client side after fetching all rows from server. At least this is what I see from H2 source code.

> If H2, how much do you think the "we just need to reverse its rows" actually costs? Nothing is free.

Since the record are ordered backward a simple iteration should suffice to swap rows i and N-i.

> Have you looked at the code which actually retrieves the rows? Do you think it is feasible (i.e., with reasonable effort/safety) to modify this code so that it actually retrieves the rows in the reversed order already, using the "inverted" index?

I think it is fairly simple to detect the inverted index. The computed Index is stored in the TableFilter. We need to set a flag for the inversion which will be used when the result are post-processed. I have not looked at the code which actually retrieves the rows. I did not consider it useful yet. I will soon at least to familiarize with it.

> How does limiting work in H2? If it uses an index to retrieve the records, is it smart enough to apply the limit before retrieving all the rows? Or does it retrieve all the rows first, then apply the limit? I understand that in the case of an ORDER BY not matching the retrieval index, it will have to retrieve all the rows and sort before applying the limit, but I am asking about the optimized case, where the retrieval index already orders the records the same as the ORDER BY.

Yes, the limit is applied on client side (LocalResultImpl.done()) after all rows were fetched (Javadoc: *This method is called after all rows have been added.*). The result is stored in a ArrayList<Value[]> rows. Which is eventually sorted (SortOrder.sort()) and trimmed (LocalResultImpl.applyOffsetAndLimit()) as specified in the query.

#13 - 10/06/2020 05:03 PM - Eric Faulhaber

Constantin, do you remember why we moved _multiplex to the front of index definitions and ORDER BY clauses? I originally had it *after* the legacy fields/columns (but before the primary key, IIRC), because I thought that having _multiplex be the leading component of every index would not differentiate the selectivity of any index for a given query, and the query planner would always have to look at the second component to choose an index anyway.

#14 - 10/06/2020 05:27 PM - Ovidiu Maxiniuc

When H2 selects the index Select.getSortIndex() it looks for the first N components of the index to be a perfect match for all N sorting criteria requested from FWD.
OTOH, adding _multiplex as sorting criterion does not make much sense as all requested/returned rows will have the same value. It only make sense that the sorting is faster as the records with other _multiplex -es are quickly dropped when H2 scans the table.

#15 - 10/06/2020 05:40 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Eric Faulhaber wrote:
>
> > Are you talking about H2 changes here, or just FWD changes?

There are two set of changes:

- one in FWD: as you see in note-4 above, the query order by tt1_3_1__i0_._multiplex asc, tt1_3_1__i0_.f1 desc nulls first will not match the (_multiplex, f1 nulls last) index. This example is taken from Constantin testcase #4785-962 - 1st commented as "this will re-sort the rows". If we want to have a match on reversed index the _multiplex sort component must be injected with the direction of the index detected from the other components.

Either this or we force the _multiplex to always be set to the same direction (e.g. asc) in both the index and the ORDER BY. Please do whichever is easier or makes the most sense to you.

- the second in H2: I do not see any attempt in org.h2.command.dml.Select.getSortIndex() to detect reversed indexes. We need to add this, or the results will not be sorted server-side. H2 will probably use some index and provide the result in that order. This is a bit of puzzle for me. We are doing our best to match the inversed index, but H2 ignores it if it cannot match a direct index and does a simple sort on client side after fetching all rows from server. At least this is what I see from H2 source code.

This seems like a pretty basic optimization that you would expect would have been done long ago. I wonder why they didn't?

> If H2, how much do you think the "we just need to reverse its rows" actually costs? Nothing is free.

> Since the record are ordered backward a simple iteration should suffice to swap rows i and N-i.

> > Have you looked at the code which actually retrieves the rows? Do you think it is feasible (i.e., with reasonable effort/safety) to modify this code so that it actually retrieves the rows in the reversed order already, using the "inverted" index?

> > I think it is fairly simple to detect the inverted index. The computed Index is stored in the TableFilter. We need to set a flag for the inversion which will be used when the result are post-processed. I have not looked at the code which actually retrieves the rows. I did not consider it useful yet. I will soon at least to familiarize with it.

If it can avoid a post-processing sort to reverse the rows, without adding noticeable cost itself, I think it is worth investigating at least.

> > How does limiting work in H2? If it uses an index to retrieve the records, is it smart enough to apply the limit before retrieving all the rows? Or does it retrieve all the rows first, then apply the limit? I understand that in the case of an ORDER BY not matching the retrieval index, it will have to retrieve all the rows and sort before applying the limit, but I am asking about the optimized case, where the retrieval index already orders the records the same as the ORDER BY.

> > Yes, the limit is applied on client side (LocalResultImpl.done()) after all rows were fetched (Javadoc: *This method is called after all rows have been added.*). The result is stored in a ArrayList<Value[]> rows. Which is eventually sorted (SortOrder.sort()) and trimmed (LocalResultImpl.applyOffsetAndLimit()) as specified in the query.

That seems like another basic optimization that you would expect from a database. If I understand you correctly, limit 1 is really not helping us here on the initial query, other than to get the results into the database cache, in case we come along with another query afterward to request them (which we would for a FOR EACH or a FIND NEXT/PREV).

#16 - 10/07/2020 01:40 AM - Constantin Asofiei

From my debugging of the AdaptiveQuery, in H2 the LIMIT is applied at the fetch, and not after re-sort, if the fetch index and the ORDER BY index match. For example, this query:

```
select
    tt1_1_1__i0_.recid as id0_, tt1_1_1__i0_._multiplex as column1_0_, tt1_1_1__i0_._errorFlag as column2_0_,
tt1_1_1__i0_._originRowid as column3_0_, tt1_1_1__i0_._errorString as column4_0_, tt1_1_1__i0_._peerRowid as c
olumn5_0_, tt1_1_1__i0_._rowState as column6_0_, tt1_1_1__i0_.f1 as f7_0_, tt1_1_1__i0_.f2 as f8_0_
from
    tt1 tt1_1_1__i0_
where
    tt1_1_1__i0_._multiplex = ? and tt1_1_1__i0_.f1 >= ?
order by
    tt1_1_1__i0_._multiplex asc, tt1_1_1__i0_.f1 asc nulls last
 limit ?
```

#17 - 10/07/2020 01:43 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, do you remember why we moved _multiplex to the front of index definitions and ORDER BY clauses? I originally had it *after* the legacy fields/columns (but before the primary key, IIRC), because I thought that having _multiplex be the leading component of every index would not differentiate the selectivity of any index for a given query, and the query planner would always have to look at the second component to choose an index anyway.

Without the _multiplex first at the ORDER BY, the WHERE will choose a different index; keep in mind that we always filter by _multiplex at the WHERE. Also, as Ovidiu mentioned, this is a performance issue, too - by using _multiplex, we filter immediately only the records for this virtual temp-table.

#18 - 10/08/2020 06:36 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Without the _multiplex first at the ORDER BY, the WHERE will choose a different index; keep in mind that we always filter by _multiplex at the WHERE. Also, as Ovidiu mentioned, this is a performance issue, too - by using _multiplex, we filter immediately only the records for this virtual temp-table.

Actually, from what I understand, the query plan will select the index based on where predicate. Since the query example from note 16 compare

multiplex and f1, the idx1 will still be chosen for table navigation but H2 have no guarantee that the order is correct. So, in case a sorting index is detected it is used and the paging (offset and count) are applied directly knowing that the sort is correct. Otherwise (no sorting index detected), H2 will filter all records matching the predicate using the index as noted above. The paging cannot be enforced because the records are not sorted. As result, a temporary buffer of 9980+ records are preselected, even if limit was 25 (FWD second stage of scroll). In the final step, those records are sorted (this reminds me of FWD client-side operations) and the requested/paged records returned. For this example, 99.7% of preselected rows are dropped when the following line is executed:

```
rows = new ArrayList<>(rows.subList(0, 25));
```

I confirm that H2 does not detect reversed indexes. Probably because it has no means to means to scan the tables in reverse order (the previous() method of IndexCursor just throws a RuntimeException)? However, the TreeCursor supports it and this is the next step of my investigation.

#19 - 10/09/2020 03:53 PM - Eric Faulhaber

Ovidiu, do you expect to have any update today? Is the first item in #4945-15 something that stands on its own?

#20 - 10/09/2020 04:27 PM - Ovidiu Maxiniuc

I will answer in reverse order.

> Is the first item in #4945-15 something that stands on its own?

Yes, I adjusted the _multiplex to be correctly match the rest of index direction.
I tested it with hotel_gui and customer code and it is stable. Probably if we use PostgreSQL as backing database for temp-tables we could see the performance increase (from my tests I noticed a ~200ms performance increase for ~2500 rows when the index is matched, in both directions).

> Do you expect to have any update today?

I can commit the _multiplex -related change-set and a set of other optimizations any moment. These are independent from H2 part. I will continue to work for a a couple of hours to get H2 to score similar results with PosthreSQL, at least for some cases, but I am not sure of the stability of my patch here and I cannot do a project-level test.

#21 - 10/09/2020 04:40 PM - Eric Faulhaber

OK, as soon as you are comfortable with your FWD-only changes, please commit them.

As to the H2 changes, please DO NOT build these on top of Adrian's update from #4949. I am not committing my PreparedStatement caching, because either that implementation or Adrian's update to H2, or some combination of the two, severely regresses performance of the "Mailinglijst" screen.

#22 - 10/09/2020 05:20 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> OK, as soon as you are comfortable with your FWD-only changes, please commit them.

Done, see r11699.

> As to the H2 changes, please DO NOT build these on top of Adrian's update from #4949. I am not committing my PreparedStatement caching, because either that implementation or Adrian's update to H2, or some combination of the two, severely regresses performance of the "M*********" screen.

OK. In fact I have not updated yet to that patch, I work with h2_1.4.200_meta_lock_fix_20200727.patch.

#23 - 10/14/2020 07:12 PM - Ovidiu Maxiniuc

I finished the solution for avoiding re-sorting the data if a reversed index is detected. I have chosen the solution with minimal changeset in order to have an easier maintenance with 3rd party code. The solution is as follows:

- detection of reversed index: after all indexes failed the sort detection the sort order is reversed and the algorithm is run again. It stops at first match and the index is stored in Select as the reversed index. Proceed to fetch records as usual;
- setup index in future result. Compare the index marked as reversed with the one used for filtering records. If we have a match, the LocalResult is notified that the data is actually sorted, even if sortUsingIndex is false;
- reorder results: finally, when the LocalResult is done, if we know the order is reversed:
  - return the last row when offset is and limit instead of scanning all result for minimal value;
  - iterate the rows and swap element i with n - i - 1 instead of sorting them before applying windowing bounds as usual.

Do we have a repository where to commit my changes?

#24 - 10/15/2020 02:50 AM - Adrian Lungu

Ovidiu, there is no H2 dedicated repository yet. My patch on H2 is not safe, so I didn't worry about a repository jut now. Please go ahead and create a repository for FWD-H2 eventually, if you are ready for commit.

#25 - 05/11/2021 06:34 PM - Ovidiu Maxiniuc

- Status changed from New to Feedback

- Priority changed from High to Normal

My changes were committed to a newly created bzr repository located at devsrv01:/opt/secure/code/p2j_repo/fwd-h2/, revision 3.