Database - Feature #7066

Implement multi-table indexed query in H2

01/23/2023 09:08 AM - Alexandru Lungu

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Radu Apetrii	**% Done:**	50%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #7061: Enable the use of lazy result sets in H2	**Closed**
Related to Database - Feature #6582: implement multi-table AdaptiveQuery	**WIP**

History

#1 - 01/23/2023 09:25 AM - Alexandru Lungu

As far as I could investigate H2, it seems like multi-table queries are severely lacking indexed traversal of joined tables to solve ORDER BY. There is a routine which checks if the selected index for traversal (by the planning phase) automatically provides the rows in the specified ORDER BY order. If it does, there is no need for a final resort of the rows, which greatly affects our FIRST / LAST queries.

Unfortunately, this routine simply checks if the ORDER BY clause is a prefix of the selected index. In multi-table queries, the ORDER BY is **always** including fields from all tables (at least multiplex and recid). Evidently, this means that the routine won't ever succeed for multi-table queries and thus all multi-table queries are resorted. This is an issue especially for #6582 where AdaptiveQuery is using ProgressiveResults (with LIMIT).

Even so, the planning phase of H2 is not that emphatic with our ORDER BY clauses. In fact, H2 rarely plans to satisfy the ORDER BY first. What is worse is that sometimes H2 actually chooses the correct indexes in regard to the ORDER BY clause, in its effort to optimize the WHERE clause, but it is not aware of that. In the end, it prefers to fetch all the records, sort them (the order is still the same, but H2 doesn't know that) and applies LIMIT.

A great deal here is the fact that FWD AdaptiveQuery is always fetching the records by an index. Basically, it is always correct to keep the joined tables order and presume the rows are already ordered when fetching. For this purpose, I think we can adjust H2 to realize that our ORDER BY is some kind of USE-INDEXES. We can generate H2 SQL like select [...] from [...] USE-INDEX cross join [...] USE-INDEX where [...] order by [...] for AdaptiveQuery. This would disallow table join permutation and will force the resolution of ORDER BY faster. Note this is greatly linked with #7063.

#2 - 01/24/2023 02:31 AM - Constantin Asofiei

Alexandru, what H2 methods are responsible for this re-sorting of the rows? I want to check in the profiler.

#3 - 01/24/2023 06:13 AM - Alexandru Lungu

You can simply check prg.h2.result.SortOrder.sort. This holds the ORDER BY clause and is triggered only when H2 needs to sort the records, mainly because the index selection didn't succeed to do it implicitly. However, you can't have a feedback on whether H2 **could** have avoided re-sorting at the end, due to better index selection.

Currently, I expect that most of the multi-table PreselectQuery (or optimized multi-table AdaptiveQuery) will show up in the profiler to be dependent

upon re-sorting.

However, this task is rather designed to aid multi-table AdaptiveQuery, because a LIMIT 1 without indexing requires the fetching of **all** records to be ultimatelly sorted. **It is not about the time wasted on sorting, but on the time wasted on retrieving all records to be sorted in order to honor LIMIT 1**.
To have a better understanding of this optimization, we should compute the total time of our multi-table LIMIT queries as all of them use resorting.

Also, this task can also help #7061.

#4 - 01/24/2023 06:13 AM - Alexandru Lungu

- Related to Feature #7061: Enable the use of lazy result sets in H2 added

#5 - 01/24/2023 06:13 AM - Alexandru Lungu

- Related to Feature #6582: implement multi-table AdaptiveQuery added

#6 - 03/07/2023 09:43 AM - Alexandru Lungu

- Assignee set to Radu Apetrii

- Status changed from New to WIP

The purpose of this task too wide, especially after the implementation of the lazy keyword. We should address some topics separately. If these are too complex, we can move on with some separate threads.

What is our current status:

- multi-table PreselectQuery is very general, as we can't rely on the fact that the ORDER BY actually matches some indexes. From this POV, we should just try our best with improving the H2 query plan. For this purpose, we should check how PostgreSQL is planning some critical queries and try to understand better what H2 lacks.
- multi-table AdaptiveQuery is quite specific, as each component can be traversed with an index to satisfy the ORDER BY. Please consider the following two:
  - FWD is generating a concatenation of sort criterions. H2 is not capable of identifying that the ORDER BY is a concatenation of sort criterions, so it pretends that is just an arbitrary sorting string. Thus, indexing won't ever help the ORDER BY in H2. I am not even sure that PostgreSQL is better on this chapter.
  - The lazy traversal of a single-table AdaptiveQuery is really good for our cause as it doesn't require server-side invalidation and full fetching. However, queries with multiple tables may be slower with lazy fetching, because of the weak planning? This is a question worth answering.
  - Supporting the invalidation of a multi-table query in H2 sounds like a nightmare. There is a complex logic regarding "referenceRecord" which should be replicated in H2. We should have an estimate of the complexity here.
- multi-table CompoundQuery is a server-side joining mechanism, so we can't help it from H2.
- **Overall, we reached a state we can't decide which is faster: iterating all joined tables in an indexed fashion to honor sorting (like 4GL does) vs using the default planning engine from H2 allowing table permutation with final resort (like H2 natively does)**

To make a step ahead, I am thinking of the following:

- For multi-table PreselectQuery, please extract some examples from large customer applications and try to compare the plans from H2 and PostgreSQL on some mock schemas. I wonder if there is some obvious optimization H2 is missing in.
- For multi-table AdaptiveQuery, do the same. We should have examples with arbitrary order-by clauses and indexed order-by clauses.
- Answer #6582-50 and compare multi-table lazy AdaptiveQuery with non-lazy AdaptiveQuery. I want to know if having lazy on multi-table is a free performance boost or a pitfall.
- Please check if we can still use USE-INDEX in H2 (I recall that at some point it was removed from H2). If so, please make some tests based on large customer application queries:
  - use USE-INDEX, but not only as a hint (ensure that the index is actually used). Suppress table permutations. Don't add an ORDER BY clause. This is the 4GL way.
  - use the current native query planner with table permutation and ORDER BY. This is the H2 way.
  - I think the best will be a middle-ground, in which we use the H2 native planner, but improve it to be more aware of the 4GL style queries.

Radu, please start addressing the matters above.

#7 - 03/29/2023 05:25 AM - Alexandru Lungu

Created 7066a from trunk rev. 14519 and rebased with 7026b changes. Also created 7066a_h2 from FWD-H2 trunk rev. 13.

#8 - 03/30/2023 06:26 AM - Radu Apetrii

I've committed to 7066a, rev. 14532 and 7066a_h2, rev. 14 some changes that target the use of use index.
As Alex pointed out in the previous posts, when dealing with a query that contains join(s), H2 computed a plan to see the order in which the tables should be iterated. Not only that, but H2 also decided to not iterate every table by its index (in some cases).

As a list of what was changed, here are the main things:

- In H2, if the statement use index is encountered, then that table will explicitly use the mentioned index. Before, H2 would only consider that index a "hint", but now it is forced to take it.
- For now, only multi-table non-invalidated AdaptiveQueries over temporary-tables benefit from the use of use index.
- In the previously described case, for a query in FWD, all the tables will benefit from the use index statement, which means that H2 will not be "playing around" with the order of the tables inside the query. Also, each table will use its own index. I expect this to show a bit of a time improvement, since H2 will not be computing all of the planning strategies.
- The ORDER BY clause from the FQL (again, in the mentioned above case) is removed (if this turns out to be a problem, things might very well change; refer to #7194).
- In fql.g and hql.g the keywords USE and REVERSE were added (if they were missing). This means that running ant jar will not be enough to satisfy the new rules.
- Because H2 did not have the feature to iterate indexes in reverse order, I've added one variable that suggests the current direction the index is facing and another one to tell if that direction should be reversed. This way, before executing the query, we know in which direction we should go.
- In H2, these changes only target TreeIndex for the moment.

Note: This solution was only lightly tested, so if there are bugs encountered, please let me know.
I'm aware that some things can be done more efficiently, and I'm planning to look into it. Also, I need to test out the actual performance improvement (hopefully this is the case).

#9 - 03/30/2023 08:27 AM - Alexandru Lungu

Review for 7066a/rev. 14532
The changes are reasonable considering the need of enforcing USE-INDEX in some scenarios. My remarks:

- s.setType(INDEX); means that you set the index_name AST type to INDEX. This way, in a USE INDEX index_name, you will have 1 AST of type USE and 2 AST of type INDEX. I think it is best to leave it be, without explicit type.
- Please change the copyright year whenever you change a file i.e. 2019-2022 to 2019-2023.
- Use another # in history entry when the entry refers to independent work (from another branch then the previous history entry). For instance, you can use 010 in AdaptiveComponent, as your change comes from other branch then the previous from 20221130.
- I like the supportUseIndex approach here.
- I think that assembleFromClause doesn't need useIndex parameter - it can use supportUseIndex() straight-away.

I see that you have several changes in FWD-H2, so I will take some time reviewing them.

#10 - 03/30/2023 09:08 AM - Radu Apetrii

Alexandru Lungu wrote:

- s.setType(INDEX); means that you set the index_name AST type to INDEX. This way, in a USE INDEX index_name, you will have 1 AST of type USE and 2 AST of type INDEX. I think it is best to leave it be, without explicit type.
- Please change the copyright year whenever you change a file i.e. 2019-2022 to 2019-2023.
- Use another # in history entry when the entry refers to independent work (from another branch then the previous history entry). For instance, you can use 010 in AdaptiveComponent, as your change comes from other branch then the previous from 20221130.
- I like the supportUseIndex approach here.
- I think that assembleFromClause doesn't need useIndex parameter - it can use supportUseIndex() straight-away.

I committed to 7066a, rev. 14533 the changes that you suggested.

#11 - 04/05/2023 05:43 AM - Alexandru Lungu

- % Done changed from 0 to 50

Review for 7066a_h2/rev. 14

Most of the changes for #7066 are conceptually inside H2. I see that there are a lot of changes regarding USE-INDEX being enforced, which I think is good for this cause. However, there is a consistent work regarding USE-INDEX **REVERSED** option, which I am a bit concerned of. Also, there are changes in regard to the already integrated lazy traversal (#7061). Radu, please make sure the REVERSED and LAZY are still what we need and still provide the performance we expect. Note that USE-INDEX can be used without LAZY in concurrent environments.

- In Select.queryFlat, I think it is best to rename nonReversedOrder to reversedOrder, especially when !nonReversedOrder is a bit confusing. Instead of doing instanceof for TreeIndex, add a getReversed method to the super-class Index. By default it returns false. Is there any case that the table index is not TreeIndex, so that the reverse mechanism won't work?
- I see you also changed the way lazy is working. Please double check customer applications, adaptive scrolling/non-scrolling tests and eventually extend you test-case suite for lazy and use-index (unless already exists). Having a good "lazy" implementation is way more important then reversed indexes.
- You also do a lot of prev implementation. **Ensure that you don't sacrifice performance just to support reversed indexes**. Worst case, we may want to stick to the old implementation and abandon implementing USE-INDEX REVERSED.
- In TreeCursor.previous, you have if (first != null && tree.compareRows(node.row, first) < 0). Shouldn't it be last instead of first? And > instead of <?
- In TreeIndex, multi-line parameter javadoc should have */ on a new empty line. Also, /* should be on an empty line. Also, do you really require two parameters needsReversing and reversed? Maybe you can just use reversed?
- In fact, **it doesn't actually make sense to have reversed index; it makes sense to have a reversed cursor**. Imagine having a concurrent environment; you don't want to mess with shared data like reversed being set and unset by different threads. I will rather invert prev with next and next with prev in reversed cursors, instead of extending implementation in Select and TreeIndex. For the planning phase, just add a new marker reversed if the index is meant to be traversed in reverse.
- In TableFilter, you replace next with prev and prev with next unconditionally. I can't tell what is the strategy here, but please be careful with such changes - do intensive testing with/without USE-INDEX and/or with/without LAZY. I don't think it is a good idea to presume that USE-INDEX is used **only** with **lazy** in single-user environments. I think it is fine to use USE-INDEX without lazy in multi-user environments like _dirty or _meta.
- I am aware that now we don't permute tables anymore. If you set the cost to 0 once USE-INDEX is used, the planner will choose the first permutation. For less than 7 tables, the first permutation is the identity one, as the planner tries all permutations. For more than 7 tables, is the first permutation still the identity one?

I will do some tests and profiling on you solution until now. The results won't be conclusive in regard to the review, but I am curious if there are obvious regressions and if the solution provides a substantial improvement in this form.

#12 - 04/05/2023 06:27 AM - Alexandru Lungu

Rebased 7066a with trunk rev. 14525. 7066a is now at rev. 14526.
Radu, please update 7066a and notify me if something is wrong after rebase. 7066a had the changes from 7026b which were merged in the meantime in trunk, so the rebasing process wasn't that straight-forward.

#13 - 04/05/2023 09:01 AM - Alexandru Lungu

Tested 7066a against a customer application: most of the tests pass, but I have one that provides an unexpected output. I can't isolate the issue; it just provides some wrong answers in the end. I will test again after you address the review in #7066. Also, please add some H2 tests to ensure the solution is sound:

- Tests with USE-INDEX in single/multi-table scenarios.
- Tests with USE-INDEX and LAZY in single/multi-table scenarios.
- I don't think that the H2 testing framework simulates multi-user environments.

Tried to profile 7066a, but encountered the same issues as in the testing phase.

UPDATE: You can allow USE-INDEX for single-table in FWD only to check the completeness. This way, you can make tests easier. You can disable it at the very end, just because H2 manages to find better plans for single-table queries.

#14 - 04/12/2023 10:23 AM - Alexandru Lungu

Radu, please make an update on #7066. AFAIK, you are planning to integrate the #7194 changes into 7066a and deliver both at once. Post here your progress so far.

#15 - 04/13/2023 08:00 AM - Radu Apetrii

Alexandru Lungu wrote:

- In Select.queryFlat, I think it is best to rename nonReversedOrder to reversedOrder, especially when !nonReversedOrder is a bit confusing. Instead of doing instanceof for TreeIndex, add a getReversed method to the super-class Index. By default it returns false. Is there any case that the table index is not TreeIndex, so that the reverse mechanism won't work?

I've refactored this as you suggested. From the tests I've ran, I have not encountered anything other than TreeIndex. If this shall change in future testing, I'll get back to this point.

- I see you also changed the way lazy is working.

I believe that initially, when lazy was first implemented, the tests I've tried did not cover these parts of the code. Now, with some of these changes, I've stumbled upon some errors, so this should actually fix it. Also, I've tested with a large customer application and all sorts of other tests (like the adaptive_scrolling ones).

- In TreeCursor.previous, you have if (first != null && tree.compareRows(node.row, first) < 0). Shouldn't it be last instead of first? And > instead of <?

You are right. I've changed this.

- Maybe you can just use reversed?

More than that, TreeIndex now uses none. It is all stored in IndexHint, and yes, only reversed is in use.

- In fact, it doesn't actually make sense to have reversed index; it makes sense to have a reversed cursor.

I've done some changes as to where this information gets stored. It is now part of the IndexHint and it's no longer in the actual index.

- I am aware that now we don't permute tables anymore. If you set the cost to 0 once USE-INDEX is used, the planner will choose the first permutation. For less than 7 tables, the first permutation is the identity one, as the planner tries all permutations. For more than 7 tables, is the first permutation still the identity one?

For a query with 8 tables, the answer is yes, the first permutation is the identity one. I can't guarantee that this will happen for any N number of tables, but it looks like it.

Additional notes:

- I've refactored the statement "use index" into "use_index" to avoid parsing issues if a table contains a column named "index".
- In H2, the class IndexHints used to allow a list of indexes to be specified. Since we explicitly tell ("force") which index to be used, then there is no need for the IndexHints to store a list. It now stores only the name of the index and a flag that marks whether things should be iterated in reverse order or not.
- In H2, in Plan.calculateCost, I've removed the if (item.cost == 0) statement because I believe that it was placed in the wrong place initially. The meaning of that if statement was to stop computing other plans if one was already found. This now happens in Optimizer.calculateBruteForceAll.
- When running build.sh testFWD for H2, there are a few (<5) tests that fail. This should not be worrying because each one of them suffers from:
  - still using "use index" rather than "use_index".
  - checking a specific plan when "use index" is used. Obviously, the index is being forced instead of just hinted so a plan might be altered because of that.
  - checking a specific order of tables when "use index" is used. With these changes, no table permutation is tried, so the result might be unexpected for H2.
- The changes from #7194 have been integrated here as they were applied in the same place in PreselectQuery.
- **I will create a set of tests that will cover all the things talked in this post. Also, I will fix the above mentioned ones so they will not result in an error.**

I've committed the changes to 7066a_h2, rev. 15 and 7066a, rev. 14527.
As soon as I finish testing with those H2 tests (which I will create) I will get back with updates.

#16 - 04/19/2023 06:34 AM - Radu Apetrii

I've committed to 7066a. rev. 14528 the following things:

- use_index is now used only in multi-table scenarios and no longer in both single and multi-table ones. If you wish to go back to both scenarios, in AdaptiveQuery.supportUseIndex there are 2 lines that need to change:
  - boolean result = components.size() > 1 && ... needs to change with components.size() >= 1.
  - if (components.size() <= 1) needs changing into if (components.size() < 1).
- lazy is now used only in single-table scenarios. I'm not sure why this wasn't always the case, but it is good to have it in place now.

More changes have happened in 7066a_h2, rev. 16:

- 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.
- I've added tests for both use_index and use_index with lazy.
- I've fixed the tests that were giving an error (more details in #7066-15).

> - In TreeCursor.previous, you have if (first != null && tree.compareRows(node.row, first) < 0). Shouldn't it be last instead of first? And > instead of <?
>
> You are right. I've changed this.

Actually, I think that only < should be changed to >. Because the direction is changed (prev instead of next), first contains the value of last (from a normal iteration). Their meaning switched the other way around.

I look forward to seeing the results of some performance tests. This way we can decide if we should extend this idea.

#17 - 05/02/2023 04:15 AM - Radu Apetrii

While doing some work on #7252, I noticed that there might be some (other) stuff that is unnecessary computed when an index is being forced to be used. Over the coming few days I will have a look and I will provide updates on my findings.

Note: This is not urgent and does not interfere with the previous commits. It is more of a check to see if a performance boost can be obtained.

#18 - 05/19/2023 06:22 AM - Radu Apetrii

While working with #7252, I found a much easier way to obtain the index name, which is done automatically instead of manually computing it myself.
Thus, I've committed this change to 7066a, rev. 14529.
If there are any more suggestions, I will follow them.

#19 - 06/30/2023 03:23 AM - Alexandru Lungu

The work here is hanging still for a while. I've done a reread of the changes here.

Currently, multi-table queries over _temp are scarce, because they are generated only by CompoundQuery.Optimizer (sometimes). Unless we get #6582, this optimization won't do much. However, #6582 alone may represent a huge performance draw-back alone, so without #7066, we can't integrate #6582. My point here is that these 2 tasks should be merged together. There is consistent work in #6582 already, but needs rebase and improvement. I can't guarantee that the final outcome will be **a huge performance improvement** and I don't know if we can approximate it at this stage.

Before doing the heavy-lifting here, I would like to check 1 thing:

- How many CompoundQuery on _temp are there and how slow they are. We can use the FWD query profiler over _temp for this.

I will do this test on a large customer application I already set-up. Radu, please do the same thing. We can merge #6582 at the right time and enable it only for _temp, until we found a better solution for persistent databases.

#20 - 06/30/2023 08:57 AM - Alexandru Lungu

Rebased 7066a and 7066a_h2 to latest FWD_H2 and FWD trunk.

#21 - 07/03/2023 04:21 AM - Radu Apetrii

Alexandru Lungu wrote:

> - How many CompoundQuery on _temp are there and how slow they are. We can use the FWD query profiler over _temp for this.

To be honest, I was expecting to find much more CompoundQueries after navigating a customer application for 15-20 minutes.
Below is a table that illustrates only the queries executed over the _temp database. Time is measured in milliseconds.

CompoundQuery	Records processed	Total time	Avg time spent per record
CQ1	1237	215.48	0.17
CQ2	4	1.67	0.42
CQ3	5	2.97	0.59
CQ3	2	0.76	0.38

Here, CQ3 is executed twice, once processing 5 records and the other time processing only 2 records.

I don't think that there are a lot of conclusions that can be drawn from this. Also, I noticed that the average times are very similar to the ones from CompoundQueries executed over a persistent database (which were a lot more).

#22 - 07/03/2023 04:59 AM - Alexandru Lungu

Hmm, I wonder if your logging also captures **multi-table AdaptiveQuery** that were optimized from CQ. My point: there are many CQ with 2/3 components that are optimized in one single multi-table AdaptiveQuery. AFAIK, your logger is targeting only CQ steps. I really wonder if the logger is hit when there is only one fat component in CQ and that is processed.

You can test this assumption outside the customer application, you shall just generate a test that triggers compound query optimization. Check if its iteration is shown in the logger.

#23 - 09/12/2023 10:30 AM - Greg Shah

Since 7066a_h2 is needed to fix a bug (regression?) that causes #7666 (and several other related tasks), what is the plan to get this into trunk?

#24 - 09/13/2023 02:59 AM - Radu Apetrii

Greg Shah wrote:

> Since 7066a_h2 is needed to fix a bug (regression?) that causes #7666 (and several other related tasks), what is the plan to get this into trunk?

The plan is to create a 7066b_h2 branch. After that, I will extract from 7066a_h2 the code that is required in order to fix those issues (#7666 and the others) and place it in the newly created branch. This way, we can get the changes into trunk without having to add all the logic from #7066.
I will try to get this done today.

#25 - 09/13/2023 04:30 AM - Radu Apetrii

Done. I committed to **7066b_h2, rev. 27** changes that target the behavior of lazy when using previous(). I tried the TestLazy file, a customer application, a test that was providing the wrong answer (before the changes) and the steps from #7666 (and the other tasks) and everything works now.

Alex, can you please take a look at the changes and check if I've missed something? Thank you!

#26 - 09/13/2023 04:46 AM - Alexandru Lungu

I am OK with the changes. The regression tests are ok. Does this also solve #7776?
Radu, **please extend your regression tests from FWD-H2 to cover this case as well**. Please let me know when you finish integrating the tests.

I will merge this as rev. 27 in trunk soon. Also, I am planning to do a full overview of other hanging FWD-H2 branches that can be merged into trunk and eventually do a "massive" testing round.
I will notify you when everything that is needed reaches FWD-H2 trunk. This way, you can do some testing with customer applications.
Hopefully we can upgrade to the latest FWD-H2 by the end of this week.

#27 - 09/13/2023 05:26 AM - Radu Apetrii

Alexandru Lungu wrote:

> I am OK with the changes. The regression tests are ok. Does this also solve #7776?

No, but it was quite straight-forward to solve that test. I committed to 7066b_h2, rev. 28.

> Radu, **please extend your regression tests from FWD-H2 to cover this case as well**. Please let me know when you finish integrating the tests.

I added tests that cover both this case and the one from #7776.

#28 - 09/13/2023 10:47 AM - Alexandru Lungu

> No, but it was quite straight-forward to solve that test. I committed to 7066b_h2, rev. 28.

Great news.

Merged 7066b_h2 into trunk as rev. 28 and archived. Radu, please address #7797 asap. Fixing it and merging in time will allow us to do a final testing round of FWD-H2 and do the upgrade.

#29 - 12/19/2023 09:50 AM - Greg Shah

What is the status of this task and of 7066a?

#30 - 12/19/2023 09:59 AM - Alexandru Lungu

It is pending mostly as it was attempting to optimize multi-table queries on H2. After some profiling and tracking, there was a low number of such queries, so this moved on a low priority. Radu may provide more insights.