Database - Feature #6995

possibly support NO-UNDO temporary tables directly in H2

12/13/2022 10:41 AM - Eric Faulhaber

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

Description

History

#1 - 12/13/2022 11:08 AM - Eric Faulhaber

- Assignee set to Alexandru Lungu

NO-UNDO temp-tables are used heavily in many 4GL applications. Typically, regular temp-tables are used as well, such that a database transaction may include changes to a mix of NO-UNDO and regular temp-tables. In FWD, the implementation of NO-UNDO temp-tables is kludgy and it stands to reason it is expensive, though we have not yet measured the specific impact.

When an UNDO event occurs, any changes made in a temp-table, whether it is marked NO-UNDO or not, are rolled back within H2, because there is no notion of a NO-UNDO temporary table in H2. So, FWD tracks all changes made during a transaction (or sub-transaction), and "manually" re-rolls these forward at the appropriate time, if they were rolled back by the database.

It seems it would be much more efficient to deal with this requirement directly in the database (i.e., not rolling back changes to a NO-UNDO temp-table in the first place), so we do not need to track these changes in the FWD runtime to "replay" them after they've been rolled back.

However, I only want to expend the effort to modify H2 in this way, if we can prove there is a significant cost in the current implementation, in real use cases. While it seems obvious that the replay mechanism is expensive, I don't in fact know how often this occurs in real use cases, and thus, its contribution to performance.

So, the first step of this task is to figure out how to measure this cost, relative to overall persistence performance. If we find the cost is a significant enough component of overall persistence costs, such that it seems worth the effort of this change to H2, we can then consider the implementation.

Alexandru, I'm assigning this to you for initial discussions, but feel free to re-assign to Danut or Radu as appropriate, once we start working out a plan.

#2 - 12/13/2022 11:13 AM - Eric Faulhaber

I guess a simple first step is to profile typical flows in real applications and see if we can associate any bottlenecks with operations in the FWD runtime (including H2) related to the tracking and replaying of changes for NO-UNDO temp-tables.

#3 - 12/13/2022 11:56 AM - Constantin Asofiei

I think Session.log is responsible for logging the sub(tx) details needed for savepoints or rollback. This is enabled only if UNDO_LOG=1, the H2's Session.undoLogEnabled boolean field, but I don't know if this is set via a JDBC URL config or something else.

Anyway, we should benchmark H2 with disabled UNDO_LOG and see what happens. FYI, for a large application, Session.log is being called ~135k times and Session.commit is being called ~75k times.

#4 - 12/14/2022 09:23 AM - Alexandru Lungu

I am aware of the force-no-undo setting which forces all tables into no-undo. If this flag is set, no SavepointManager is used. Therefore, your scenario is only for those cases where force-no-undo is not used, right?

Therefore, this optimization targets applications which do not use such option. In this case, I will do some preliminary tests on customer applications removing force-no-undo. I will check how much of the time is spent into roll-back + re-roll for NO-UNDO tables.

> I think Session.log is responsible for logging the sub(tx) details needed for savepoints or rollback. This is enabled only if UNDO_LOG=1, the H2's Session.undoLogEnabled boolean field, but I don't know if this is set via a JDBC URL config or something else.

It is part of the JDBC URL config and we do UNDO_LOG=0 when using force-no-undo. By the way, this is removed in the latest H2 versions, because it is specific to PageStore.

> Anyway, we should benchmark H2 with disabled UNDO_LOG and see what happens. FYI, for a large application, Session.log is being called ~135k times and Session.commit is being called ~75k times.

I guess we need a more granular statistic: how many times a certain type of table (NO-UNDO) is rolled back. I will add a dummy keyword ("NO-UNDO") to H2 and extend the logger to make a distinction between UNDO and NO-UNDO.

#5 - 01/04/2023 05:14 AM - Alexandru Lungu

- Status changed from New to WIP

- % Done changed from 0 to 10

I've added a OrmRedo Timer JMX in 3821c/rev. 14469. I've debugged some local tests to find out a "pattern" scenario which may trigger "redo"; something like the following should hold. By this, I wanted to stress-out the particularity of such case and a possible motivation for the following results.

- code should be executed in a logical 4GL subtransaction.
- use a buffer for an undoable table and a buffer for a no-undo table
- release the no-undo buffer to allow the changes on the no-undoable DMO to be flushed. This way we are forced to redo the changes.
- release the undo buffer to allow the change of the undoable DMO to be flushed. This way it makes sense to go back to a savepoint and not simply doing an in-memory rollback.
- trigger an undo event: this way the changes on the no-undo record which are now in the database are rollbacked due to a savepoint rollback

(needed for the undoable flushed records). This requires a redo.

I tested with a large customer application (~10 mins POC) and I got only 7 redo operations solved in ~20ms in total (lets say ~3ms on average). Even so, that 7 redo ops happened only when I got some logical errors in the application; none of them were spontaneous. This makes sense as the application ensured data consistency by undoing.

Furthermore, I just added OrmNoUndoTrace in 3821c/rev. 14470. This is used to analyze the overhead of creating SQLRedo operations and storing them in case a redo ever happens. Some of the operations are using BaseRecord.deepCopy. The results for the same scenario as before leads to ~20000 redo operations created and stored in ~60ms (0.003 ms per operation).

I feel like this kind of tracking may provide different results for different applications. I will continue the tracking on other applications.

#6 - 01/06/2023 01:38 AM - Eric Faulhaber

Constantin, do you see measurable SQLRedo and/or SavepointManager activity in your profiling of the large application POC? SQLRedo indicates NO-UNDO temp-table work we could eliminate. SavepointManager is less specific, as it could represent regular or NO-UNDO temp-tables, or persistent tables.

#7 - 01/06/2023 02:02 AM - Eric Faulhaber

Alexandru Lungu wrote:

> I tested with a large customer application (~10 mins POC) and I got only 7 redo operations solved in ~20ms in total (lets say ~3ms on average). Even so, that 7 redo ops happened only when I got some logical errors in the application; none of them were spontaneous. This makes sense as the application ensured data consistency by undoing.

This metric alone suggests to me this is not a high priority candidate for performance improvement.

> Furthermore, I just added OrmNoUndoTrace in 3821c/rev. 14470. This is used to analyze the overhead of creating SQLRedo operations and storing them in case a redo ever happens. Some of the operations are using BaseRecord.deepCopy. The results for the same scenario as before leads to ~20000 redo operations created and stored in ~60ms (0.003 ms per operation).

This finding is less clear to me. Would you consider this a bottleneck in the context of your hardware, relative to the timings of other operations?

#8 - 01/06/2023 02:17 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, do you see measurable SQLRedo and/or SavepointManager activity in your profiling of the large application POC? SQLRedo indicates NO-UNDO temp-table work we could eliminate. SavepointManager is less specific, as it could represent regular or NO-UNDO temp-tables, or persistent tables.

SQLRedo is minimal - no impact. Same for SavepointManager.rollback. Only SavepointManager.registerBlockHooks has a count of 66k, and it creates some 150k objects.

#9 - 01/06/2023 02:29 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric Faulhaber wrote:
>
> > Constantin, do you see measurable SQLRedo and/or SavepointManager activity in your profiling of the large application POC? SQLRedo indicates NO-UNDO temp-table work we could eliminate. SavepointManager is less specific, as it could represent regular or NO-UNDO temp-tables, or persistent tables.
>
> SQLRedo is minimal - no impact. Same for SavepointManager.rollback.

Another data point that suggests this is not a high priority candidate.

> Only SavepointManager.registerBlockHooks has a count of 66k, and it creates some 150k objects.

In a mixed environment of regular and NO-UNDO temp-tables, this needs to happen regardless of whether we modify H2 to support NO-UNDO temp-tables natively. Well, something like this needs to happen; there may be a way to optimize it.

In any case, I'd like to get Alexandru's response to #6995-7 before we decide to put this H2 change on the back burner.

#10 - 01/06/2023 05:09 AM - Alexandru Lungu

Eric Faulhaber wrote:

> Furthermore, I just added OrmNoUndoTrace in 3821c/rev. 14470. This is used to analyze the overhead of creating SQLRedo operations and storing them in case a redo ever happens. Some of the operations are using BaseRecord.deepCopy. The results for the same scenario as before leads to ~20000 redo operations created and stored in ~60ms (0.003 ms per operation).

> This finding is less clear to me. Would you consider this a bottleneck in the context of your hardware, relative to the timings of other operations?

Hard to say. The SQLRedo instances are created and stored pretty fast for the profiled scenario IMO. I am rather worried about the high number of instances created in the process: ~20k SqlRedo, most of which store a deep copy of a DMO. There may be an impact on the GC that I didn't profiled. Implementing H2 level NO-UNDO removes overhead of storing these operations and deep copies.

Note that these times are highly dependent upon the record size, operation type and frequency. INSERT and UPDATE more precisely use deepCopy (instance creation using reflection, array copy and some bitset work).

#11 - 01/06/2023 08:17 AM - Greg Shah

If we are doing any substantial work implementing NO-UNDO in the customer scenario, then let's move ahead.

#13 - 01/26/2023 08:21 AM - Radu Apetrii

- File fwd-h2-6995.patch added

- File 6995-6129b.patch added

I've included in H2 a flag for temporary tables that shows whether or not a table is defined with no-undo. The main changes to the structure of the project are:

- When generating the SQL that will be sent to H2, if the table is defined with no-undo, then the keyword noundo now appears at the end of the SQL, before the delimiter.
- In H2, when storing the changes done to a record (in Session.undoLog), if the respective table is defined with no-undo, then no change is stored.
- Continuing from the last point, when an undo operation happens, nothing would be done to the records of the no-undo table.
- Redo operation is completely removed, as it is not necessary anymore.

There are 2 patches included, one for the FWD code and one for the H2 code. I used 6129b, rev. 14382 and fwd-h2, rev. 8. Also, these changes were tested with a customer application.
Note: The patch for fwd-h2 includes a small fix from #6928-21.
If you encounter any problems, let me know.

#14 - 01/31/2023 03:49 AM - Alexandru Lungu

Radu, based on our new FWD-H2 branching system #7058, I've created 6995a_h2 to hold your patch from #6995-13. However, I can't trivially use the patch due to a conflict in src/main/org/h2/command/dml/Update.java. I see that the patch also includes your changes from #6928 (which I think are safe to remove as rev. 6 doesn't have problems with UPDATE).
Please check-out 6995a_h2, redo your patch for rev. 6, apply it over the branch, run ./build.sh test and note my remarks from #7058-6 regarding tests. I am planning to do some performance tests with your changes asap today.

#15 - 01/31/2023 05:06 AM - Radu Apetrii

I committed to 6995a_h2, rev.7 the changes that you asked for.
Also, when running build.sh test, the errors were the same as in #7058-6, so I hope that you won't encounter any problems. As always, let me know if this is not the case.

#16 - 01/31/2023 08:27 AM - Alexandru Lungu

- Assignee changed from Alexandru Lungu to Radu Apetrii

- % Done changed from 10 to 70

I tested 6995a_h2 with your FWD patch against a large customer application and there was no regression.
The performance increase, as expected after #6995-5, wasn't groundbreaking (-0.4% from the total execution time of my test). However, we can agree that this is a performance increase for us to take.

Review of #6995-13
I am OK with the changes inside H2.
In FWD I have the following remarks:

- please reuse getCreateTemporaryTablePostfix with a no-undo flag, rather than implementing a new getCreateTemporaryTableNoUndoPostfix
- I am not necessarily sensitive with huge chunks of code being removed (less is faster), but should we keep some of the old code for reference? Removing most of SqlRedo and SavepointManager code makes the FWD code-base completely incompatible with any other H2 or FWD-H2 version. Therefore, we should take care of upgrading FWD in the same time with H2. As a matter of fact, I think we are going to have several such cut-points in our development taking into account the increasing work on FWD-H2.

I am looking forward for Eric's opinion on this one.

#17 - 01/31/2023 08:36 AM - Alexandru Lungu

Eric, I added you as a watcher now.

#18 - 01/31/2023 08:59 AM - Greg Shah

> I am not necessarily sensitive with huge chunks of code being removed (less is faster), but should we keep some of the old code for reference? Removing most of SqlRedo and SavepointManager code makes the FWD code-base completely incompatible with any other H2 or FWD-H2 version.

My opinion is that we should not clutter the code with dead stuff.

- Dead code has a real cost. Everyone looking at the code will spend more time on it than needed because there is dead code there. This multiplies over the lifetime of the code, so it can be very expensive.
- We are moving aggressively here to modify H2 and it will be an essential part of our performance solution. We are not going back. Either the H2 maintainers will accept our changes or not. If not, then we maintain our own fork of the project.

Eric has the final say here.

#19 - 01/31/2023 11:25 AM - Eric Faulhaber

I have not yet reviewed the changes in detail, but my opinion is that removing all SQL redo code from FWD is OK as long as:

- we have the requisite NO-UNDO support within H2 to support that change in FWD; and
- the pair of change sets in FWD and H2 are clearly documented as dependent upon each other and must happen in lockstep; and
- the set of changes in both are scoped only to this issue (i.e., are not intermingled with any larger set of changes).

That way, we can always see in version control what the exact set of changes were to enable this shift.

#20 - 02/01/2023 09:49 AM - Alexandru Lungu

Eric Faulhaber wrote:

> I have not yet reviewed the changes in detail, but my opinion is that removing all SQL redo code from FWD is OK as long as:
>
> - we have the requisite NO-UNDO support within H2 to support that change in FWD; and
> - the pair of change sets in FWD and H2 are clearly documented as dependent upon each other and must happen in lockstep; and
> - the set of changes in both are scoped only to this issue (i.e., are not intermingled with any larger set of changes).
>
> That way, we can always see in version control what the exact set of changes were to enable this shift.

I agree. Other changes we have for H2 follow the same patters: upgrade FWD with its corresponding H2. My concern is mainly the fact that H2 is not synchronized with FWD "out-of-the-box". Reverting/upgrading to an older/newer FWD means also reverting/upgrading to an older H2 and vice-versa.

Right now, H2 is delivered through GCD maven repository and is currently set at revision 6 (trunk). Because there are several changes/branches of H2 (each one with an artifact), should we expect to pull only trunk H2 builds from the GCD maven repository? If we are to test with a H2 branch, I guess we can just check-out and build. However, we may need to find a way to tell gradle to use a local artifact of H2 instead of the online one. If this is something we need to discuss in depth, I think we can move to #7058.

#21 - 02/07/2023 06:34 AM - Alexandru Lungu

Radu, please redo the FWD patch considering my first comment in #6995-16 regarding getCreateTemporaryTablePostfix over the new 6129c. Hopefully, the re-patching won't be that complex.

Eric, do you plan to review the FWD/H2 changes from #6995-13? Once 6129c reaches a stable state, we can go with the latest plan from #7058-11.

#22 - 02/08/2023 03:41 AM - Eric Faulhaber

Code review 6995-6129b.patch and fwd-h2-6995.patch:

Nice work, Radu!

I agree with Alexandru's comments in #6995-16 and as noted previously, I am ok removing all the SQL redo code, under the conditions stated in #6995-19. In fact, I am happy to be rid of all the redo baggage; that was a very awkward requirement that I never liked.

The rest of the changes look fine to me. I am more familiar with the FWD code than the H2 code, so I am relying on your and Alexandru's experience in this area.

#23 - 02/08/2023 09:46 AM - Radu Apetrii

- File 6995-6129c.patch added

Alexandru Lungu wrote:

> Radu, please redo the FWD patch considering my first comment in #6995-16 regarding getCreateTemporaryTablePostfix over the new 6129c. Hopefully, the re-patching won't be that complex.

Done. Moved the patch to 6129c and refactored the getCreateTemporaryTablePostfix function so it now contains a parameter. This was done with 6129c, rev. 14804.
If you feel like more documentation is needed, please let me know.

#24 - 02/09/2023 10:54 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 70 to 100

Rebased from trunk and merged 6995a_h2 to FWD-H2 trunk: includes the addition of a NO-UNDO keyword being natively supported in H2.

Once we have a new build of FWD-H2, we can also integrate the FWD patch.

#25 - 02/27/2023 08:53 AM - Alexandru Lungu

Integrated the patch into 7062a. Also, the H2 changes made it to trunk.

I've reviewed the changes again and I have a concern:

- Can't we have both transactional and no-undo create table statements? I see that "transactional" is applied only for !noUndo.
- Please note that transactional doesn't mean that the DML operations over the table are logged to be able to commit/rollback - this is by default. transactional is a keyword added by the FWD team a while ago to make CREATE TABLE a "transactional statement". In the original H2, creating a table commits the existing transaction, creates the table and opens a new one for the upcoming changes. Adding transactional boost the performance to avoid the transaction open/close overhead.

Please consider making "transactional" available for both undo and no-undo tables (in H2 dialect). Also test the H2 parser to be able to parse both transactional and noundo.

UPDATE: Also remove the JMX we no longer use for REDO operation.

#26 - 03/02/2023 04:49 AM - Radu Apetrii

Alexandru Lungu wrote:

- Can't we have both transactional and no-undo create table statements? I see that "transactional" is applied only for !noUndo.

I double checked and we can have both. In that case, the function gets called twice, but with different parameters (this is the current manner).

> *UPDATE*: Also remove the JMX we no longer use for REDO operation.

Done. Committed this to 7026a, rev. 14508.

#27 - 03/02/2023 05:12 AM - Alexandru Lungu

- Status changed from Review to WIP

- % Done changed from 100 to 80

- deleted -

Wrong post, it was meant to be for #7061.

#28 - 03/02/2023 05:13 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 80 to 100

#29 - 03/15/2023 11:10 AM - Alexandru Lungu

- Status changed from Review to Test

This reached trunk with 7026a. It can be closed.

#30 - 04/07/2023 02:10 AM - Eric Faulhaber

- Status changed from Test to Closed

Files

6995-6129b.patch	25.2 KB	01/26/2023	Radu Apetrii

fwd-h2-6995.patch	5.52 KB	01/26/2023	Radu Apetrii
6995-6129c.patch	29.6 KB	02/08/2023	Radu Apetrii