Database - Feature #7323

Implement soft unique index in FWD-H2

05/03/2023 10:48 AM - Alexandru Lungu

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

Description

History

#1 - 05/03/2023 11:30 AM - Alexandru Lungu

I've done some logging with FWD-H2 on a large customer application and identified the following for a POC run. Out of ~100k SELECT statements, almost 50% of them are in fact unique index checking. Most of the other queries are lazy or find queries. I didn't include in this counting the direct-access queries.

By unique-index checking queries, I mean something like: select 0 from tt3807 where recid != ? and _multiplex = ? and some_field = ? generated by Validation.validateUniqueByQuery. Some of them are actually using "union all" to check two unique indexes at once.

Each time I am seeing the following pattern:

```
select 0 from tt3807 where recid != ? and _multiplex = ? and some_field = ?;
-- eventually a savepoint here
insert into tt3807 (...) values (...);
commit; -- or rollback
```

Having more control into the FWD-H2 engine, I am thinking of solving this faster.

- First idea is to use direct-access to solve the selects. We already have getByUniqueIndex which does a look-up on an unique index. This will fairly cut the time of selects by 60% (due to the tests in #7062).
- Another neat approach is to extend FWD-H2 with some special syntax exactly for this case. I am considering:
  - CREATE SOFT-UNIQUE INDEX, which is basically a simple index which does uniqueness checks only when explicitly stated by the INSERT or UPDATE.
  - INSERT INTO [...] VALIDATE and UPDATE [...] VALIDATE, which triggers the uniqueness checks of the soft-unique indexes.
  - In case the uniqueness check fails, an exception is thrown back to the FWD engine which can properly transform it into a validation exception.

This way we avoid the additional index traversal / parsing / caching / planning of the select statement. Also, this technique covers the case of multiple unique indexes smoothly.

I am aware for some time that we don't use UNIQUE INDEX in our FWD-H2 databases and thus I came up with this SOFT-UNIQUE INDEX solution which gives us control on when we need to check or not the uniqueness. However, I am not sure which are the exact cases in which we don't want to use VALIDATE. From the statements I extracted, maybe all ~50k unique constraint selects are right before an insert. Please advice.

#2 - 05/05/2023 10:11 AM - Alexandru Lungu

I just realized that direct-access is not necessarily an option because these validation statements are prepared only once per table and cached. Basically, the direct access will optimize only ~4k SELECT statements out of all ~50k and only with 60% degree (empirically, this is the usual parsing and planning time for _temp tables). I don't expect more than 10ms on this part.

However, all ~50k SELECT statements are doing the index traversal work. Therefore, I would go ahead with merging the insert and validation select.

Eric, Constantin, Ovidiu: is the lack of H2 unique indexes for temporary tables out of functional reasons? Or maybe it was considered that doing a prior simple SELECT is faster than a SAVEPOINT to avoid partial inserts in case of constraint violation. I agree that the dirty database, for instance, doesn't require such constraints. However, my case of ~50k validation SELECTS (50% of them with UNION ALL) are over temporary tables (in-memory, embedded, single-user, lock-mode 0). Now that we have more control over H2, we can integrate this constraint violation check where/how we want.

- My point was using SOFT-UNIQUE / VALIDATE checks, just to control when the unique check is done. This means that we can mark legacy UNIQUE indexes as SOFT, and use INSERT INTO [...] VALIDATE whenever we feel like we should do the validation. However, I would go with such solution only if we have cases of both invalidating/validating inserts.
- If this unique check should be done always (just like in persistent databases), then we can use UNIQUE and work our way into FWD-H2. I also feel like such information of uniqueness can help the planner and performance overall. Again, this only for _temp database.

#3 - 05/05/2023 10:19 AM - Alexandru Lungu

Alexandru Lungu wrote:

> prior simple SELECT is faster than a SAVEPOINT to avoid partial inserts in case of constraint violation.

Just looked in FWD-H2 and the unique constraint violation reverts, so it doesn't end up in partial inserts. H2 removes the row from the already updated indexes. Basically, doing a failing insert is working as expected in FWD-H2; there is not even a need for a SAVEPOINT, as FWD-H2 is doing the reverting job on its own. At this point, I am uncertain of the reasoning behind the lack of unique indexes for _temp H2.

#4 - 05/08/2023 03:30 PM - Eric Faulhaber

Alexandru Lungu wrote:

> **Eric, Constantin, Ovidiu**: is the lack of H2 unique indexes for temporary tables out of functional reasons? Or maybe it was considered that doing a prior simple SELECT is faster than a SAVEPOINT to avoid partial inserts in case of constraint violation. I agree that the dirty database, for instance, doesn't require such constraints. However, my case of ~50k validation SELECTS (50% of them with UNION ALL) are over temporary tables (in-memory, embedded, single-user, lock-mode 0). Now that we have more control over H2, we can integrate this constraint violation check where/how we want.

The lack of H2 unique indices for temp-tables exists so that we can mimic the behavior of the 4GL, whereby a newly created record in one buffer can be seen by a query using a different buffer, even before the fields which would make the new record unique are fully updated. This requires that we flush the new record even before it is properly validated to conform with unique constraints. We use the RecordNursery for this early flush of an

"unfinished" new record. This would not work in some cases, if a unique index existed. Later, when we determine the record is "finished" and ready to be flushed, we go through the normal validation, and we expect to catch any unique constraint violation using the unique check query at that time. This timing should correspond with the 4GL's timing, such that any unique constraint error condition is reported at the correct point in the program.

The alternative (at least before we started heavily modifying H2 internals) was to use the dirty database (even though it was not a cross-session use case) to perform cross-buffer access of these "unfinished" new records. But this proved too heavyweight and only worked with converted FIND statements.

- My point was using SOFT-UNIQUE / VALIDATE checks, just to control when the unique check is done. This means that we can mark legacy UNIQUE indexes as SOFT, and use INSERT INTO [...] VALIDATE whenever we feel like we should do the validation. However, I would go with such solution only if we have cases of both invalidating/validating inserts.
- If this unique check should be done always (just like in persistent databases), then we can use UNIQUE and work our way into FWD-H2. I also feel like such information of uniqueness can help the planner and performance overall. Again, this only for _temp database.

This is an interesting approach that I think is worth investigating. We would have to be sure that the insert is only done when the record should be flushed, however (i.e., there are cases when a record only is validated, but not flushed/inserted). As you note, this would be for temp-tables only, so the validation logic would need to be refactored to take advantage of this optimization in H2, while still working the "old" way for persistent databases.

The reason for the separate unique constraint query check is to control when and how a legacy validation error is detected and reported. If we had not implemented this way and instead relied upon unique constraints at the database reporting a violation, we would foul the JDBC connection and be unable to continue working. This comes at the cost of a separate query (indeed, possibly several queries UNION'ed together) to perform the unique constraint check. The use of UNION was done with the assumption that in the case where an update required multiple unique indices to be checked, the lesser of two evils was to execute multiple queries and union the results together, rather than potentially taking multiple round trips to the database.

#5 - 05/15/2023 05:23 AM - Alexandru Lungu

- % Done changed from 0 to 30

Eric Faulhaber wrote:

> The lack of H2 unique indices for temp-tables exists so that we can mimic the behavior of the 4GL, whereby a newly created record in one buffer can be seen by a query using a different buffer, even before the fields which would make the new record unique are fully updated. This requires that we flush the new record even before it is properly validated to conform with unique constraints. We use the RecordNursery for this early flush of an "unfinished" new record. This would not work in some cases, if a unique index existed. Later, when we determine the record is "finished" and ready to be flushed, we go through the normal validation, and we expect to catch any unique constraint violation using the unique check query at that time. This timing should correspond with the 4GL's timing, such that any unique constraint error condition is reported at the correct point in the program.
>
> The alternative (at least before we started heavily modifying H2 internals) was to use the dirty database (even though it was not a cross-session use case) to perform cross-buffer access of these "unfinished" new records. But this proved too heavyweight and only worked with converted FIND statements.

I was expecting this; however I see the following code in Validation:

```
if (multiplex != null || !flush)
{
   // validate by executing unique index queries
   validateUniqueByQuery(check);
}

if (flush || buffer.isAutoCommit())
{
   // validation will be done by the database as part of the flush (for persistent tables)
   flush();
}
```

isAutoCommit is false for RecordBuffer and is !persistenceContext.isTransactionOpen() for TemporaryBuffer.
I see 4 cases here:

- persistent DMO / no-flush : triggers validation, but it is not flushed
- temp DMO / no-flush: triggers validation, it is flushed only if !persistenceContext.isTransactionOpen()
- persistent DMO / flush : flushed and validation is done by the unique index in the database
- temp DMO / flush : triggers validation and flushed

I am thinking to target only the temp DMO / flush case. Basically, if we are going to flush a temporary DMO (flush == true), then do it using VALIDATE and avoid doing the validateUniqueByQuery pre-check. We will catch the validation issue using a ValidationException.

> The reason for the separate unique constraint query check is to control when and how a legacy validation error is detected and reported. If we had not implemented this way and instead relied upon unique constraints at the database reporting a violation, we would foul the JDBC connection and be unable to continue working. This comes at the cost of a separate query (indeed, possibly several queries UNION'ed together) to perform the unique constraint check. The use of UNION was done with the assumption that in the case where an update required multiple unique indices to be checked, the lesser of two evils was to execute multiple queries and union the results together, rather than potentially taking multiple round trips to the database.

Corrupting the JDBC connection (especially in H2) is no limitation due to our customization possibilities. We can work our way into preserving the _temp H2 state while correctly doing the validation.

#6 - 05/26/2023 06:21 AM - Alexandru Lungu

Radu, I created 7323a_h2 and 7323a.

Please implement SOFT_UNIQUE and VALIDATE keywords. Focus first on:

- Syntactic changes adding these two to the FWD-H2 parser.
- FWD changes to emit these (without suppressing any current behavior).

At that point, we (you and I) can test how many such VALIDATE queries are generated vs non-VALIDATE. I really expect a high ratio, such that we decide to use VALIDATE instead of server-side validation when possible.

If the ratio is high enough, mark it as top priority and:

- Implement the H2 logic for SOFT_UNIQUE and VALIDATE + add FWD-H2 tests for such syntax.
- Suppress the FWD validation for queries with VALIDATE. Handle in FWD failing validation. Make extensive testing in FWD.

#7 - 05/31/2023 07:17 AM - Radu Apetrii

I've added the SOFT_UNIQUE and VALIDATE keywords to both FWD and FWD-H2 without affecting the current behavior.
On a large customer application, **80% of the queries executed over a temporary database resulted in VALIDATE being generated.**

Unless you have something to add, I will move on with the implementation of this feature.

#8 - 05/31/2023 07:27 AM - Alexandru Lungu

This looks OK to me, go ahead with the implementation.
Please do a second check for the flush method. Is there any chance flush won't execute the VALIDATE statement? So that we skip validation **and** have a non-validating flush?

#9 - 06/01/2023 04:47 AM - Alexandru Lungu

Radu, please commit this changes to 7323a. I would like to do a second test on a customer application. Also, please let me know how you've done the tracking (JMX / simple counter / debug injection).

#10 - 06/01/2023 07:08 AM - Radu Apetrii

I've committed to 7323a, rev. 14587, and 7323a_h2, rev. 21.
These changes target the addition of VALIDATE and SOFT_UNIQUE keywords and contain the syntactic changes as well as the implementation.

No issues were found while testing on a large customer application. **After ~15 minutes of testing, I encountered ~70.000 queries that contained the VALIDATE keyword.**

Note: In 7323a_h2, in ParserUtil, I also changed the position of LAZY keyword so that it matches the alphabetical order of the other keywords.

Alexandru Lungu wrote:

> Please do a second check for the flush method. Is there any chance flush won't execute the VALIDATE statement? So that we skip validation **and** have a non-validating flush?

I have double checked and I am pretty confident that this case will never happen.

I would like to do a second test on a customer application. Also, please let me know how you've done the tracking (JMX / simple counter / debug injection).

There are two types of results, so let me describe the processes below:

- To get to that 80% mark, I first considered only queries executed over a temporary database. After that, I printed with System.out.println the expression flush || buffer.isAutoCommit(). Then I counted how many true s I got and how many false s and computed the percentage.
- To get ~70.000 queries with VALIDATE keyword in them, I printed all the UPDATE and INSERT statements that were executed with/in FWD-H2. After that, I counted how many of them contained VALIDATE.

Let me know if there is something else to add.

#11 - 06/06/2023 07:55 AM - Alexandru Lungu

- % Done changed from 30 to 70

- Assignee changed from Alexandru Lungu to Radu Apetrii

I am checking out this right now.

I will run some tests/profiling on my own and keep you updated.

Anyway, I feel we should check if validate fails are actually handled correctly. You checked a large customer application in which (most probably) all validations were alright. You should also do some tests where validations are **not** alright. Try violating an unique index with temp tables - check if the fallback is done right:

- that the second record is not inserted / updated
- the first record is not altered
- H2 is in a valid state, so you can continue creating / updating new records.
- check UNDO and NO-UNDO, check TRANSACTION / SUB-TRANSACTION, etc.

Basically, we should double or triple check that unique index violation does a nice fallback.

#12 - 06/06/2023 08:41 AM - Alexandru Lungu

Review to 7323a, rev. 14587

- I don't think we should always allow soft_unique right from Dialect. Better extend this behavior in P2JH2Dialect; Dialect will throw an exception if soft_unique is requested. P2JH2Dialect will do the right append.
- I think it is better to stick with two methods: getCreateIndexString with 1 parameter and getCreateIndexString with 2 parameter. Let only the places where H2 _temp is used to consume the second interface. I don't want to let this "soft_unique" parameter be requested all over the place. It is not that intuitive, so many teammates will ask "what is soft unique, should it be true or false?".
- The same goes for other methods like flush, insert or update. No need to raise any confusion for further uses.
- I don't see any change to the catch SQLException in case of validation problems. In regard to my last comment, this should be tested in-depth.

Review to 7323a_h2, rev. 21

- Most of the changes make sense; good job!
- I see lot of getValidateMode() in dml; can you use a variable directly for faster access? (validateMode).

- Please make some unit tests for this. I want to see them in action for any environment we test (in-memory, persistent, remote). I expect to see tests from both categories: which fail / don't fail the unique check.
- Also note that the regressions test don't pass. It seems that there are parsing issues?

Don't forget update the H2 fork wiki with the new syntax addition (after we merge FWD-H2 in trunk - hopefully we won't miss this out).

#13 - 06/13/2023 06:14 AM - Alexandru Lungu

Radu, please focus on this after you finish the recent performance logging changes. I will like to give it a performance trial by the end of this week. This is really close to stable - no need to delay it further.

#14 - 06/19/2023 08:18 AM - Radu Apetrii

I've applied the changes suggested in #7323-12 for both 7323a and 7323a_h2.
I've also added tests in 7323a_h2 and fixed the fact that regression tests didn't pass (the keywords SOFT_UNIQUE and VALIDATE were missing from the list of keywords in Parser).
The changes can be found on 7323a, rev. 14588 and 7323a_h2, rev. 22.

Also note that I have tested some examples that violate an unique index and the fallback outcome is identical to the one before the changes from this task.

#15 - 06/23/2023 03:48 AM - Alexandru Lungu

- % Done changed from 70 to 100

- Status changed from WIP to Review

Review of 7323a, rev. 14588

- The changes are OK.
- Please add some extra javadoc to Persister.insert to better understand what validateMode is about. Mention that this is intended only to trigger soft index if any. Specify that this is supported only for _temp now and should be set on true only when we need to let H2 do the validation for us. Please do your best do document all places you added validateMode to give more context.

Review of 7323a_h2, rev. 22.

- The changes are OK.

As there are no function issues from my POV, I will start a testing / profiling round. I will keep you updated with the progress.

#16 - 06/23/2023 06:00 AM - Radu Apetrii

Alexandru Lungu wrote:

> - Please add some extra javadoc to Persister.insert to better understand what validateMode is about. Mention that this is intended only to trigger soft index if any. Specify that this is supported only for _temp now and should be set on true only when we need to let H2 do the validation for us. Please do your best do document all places you added validateMode to give more context.

I've added some extra javadoc in the places where validateMode is used and I've committed to 7323a, rev. 14589.

#17 - 06/23/2023 06:59 AM - Alexandru Lungu

The profiling results are quite interesting. The average improvement is only -0.2%. **However**, I've got a run with -2% (reaching the best time I ever had on my profiling tests) and one with -0.5%. Also, I didn't have one single run slower than the baseline. I really think this has potential, but still needs some attention. I will redo a test regarding how many VALIDATES were actually generated in my case. Radu, do you have an idea how we can optimize the following?

```
ps[i] = conn.prepareStatement(insertSql[i] + " validate");
```

The SQL here is not interned. AFAIK the prepared statement cache will have to equals each such validate statement, instead of reference checking (=) the precomputed insertSql (+ compute hash). The alternative is to keep an insertValidateSql array, but hopefully this won't make the memory go crazy. Please precompute insertSql[i] + " validate" in a separate array, so that we can benefit from string equality and one-time hashing. Otherwise, I am open to alternatives.

#18 - 06/23/2023 09:46 AM - Radu Apetrii

I've implemented the logic for insertValidateSql and tested with a customer application. The changes can be found on 7323a, rev. 14590.

Alexandru Lungu wrote:

> Otherwise, I am open to alternatives.

I don't have other ideas at the moment. I think we should give this approach a shot until we find a better one.
Anyway, it seems to me that we got the better end of the deal - the performance improvement should be more visible than the memory increase.

If other methods come to my mind, I will let you know.

#19 - 06/26/2023 03:13 AM - Alexandru Lungu

Review of 7323a, rev. 14590

- This is OK.
- My only concern is related to validatedMode ?. I think you should do a warning logging and fallback to insertSql[i] if validateMode is true and insertValidateSql[i] is null. Otherwise, a misconfiguration here (e.g. validateMode on true for a persistent database) may lead to NPE, which is quite fatal.

#20 - 06/27/2023 05:19 AM - Alexandru Lungu

Quite concerning, this runs slower by +0.5%. Only 2/5 profiling runs were faster than the baseline, but with just a bit.

However, I just noticed that there are some exceptions thrown, so this slow down may be caused by these exceptions. I must admit that I didn't checked the server log in #7323-17.
Could you see any exception being thrown when testing with a customer application? Error during insert
org.h2.jdbc.JdbcSQLIntegrityConstraintViolationException

#21 - 06/27/2023 05:36 AM - Alexandru Lungu

- % Done changed from 100 to 70

- Status changed from Review to WIP

Radu, please check the following hypothesis from Validation.validateUniqueByQuery:

```
// violated indices are detected in P4GL legacy order; for each one, confirm there is no
// modified DMO in memory which cures the violation
```

It looks to me that we already do some kind of recovery in case the validation fails, using Validation.confirmUniqueConstraintViolation.

There are two things I think you should investigate thoroughly:

- I think that FWD should interpret the "validateMode" a bit differently. It seems to me that queries over persistent are conceptually validating right? (as they throw errors when unique is violated, **so they validate**). It is counter-intuitive to have validateMode false, but the database to do the validation anyway. It is for the best to change the semantics of validateMode a bit. Instead of "append VALIDATE at the end", it should mean "allow database to check uniqueness". Of course, if validateMode is false and a persistent database is used, this reaches an invalid state. Radu, consider the right changes to transform validateMode into a allowDBUniqueCheck. Databases that can't suppress unique checks will log a SEVERE, but still do the insert/update.
- Please consider confirmUniqueConstraintViolation as this was always happening for _temp tables before. With your changes, confirmUniqueConstraintViolation is no longer used - maybe move it after you catch the unique violation to recover - but only when multiplex != null. In fact, do some analysis on confirmUniqueConstraintViolation to better understand its goal. I just skimmed the code there. Be sure you keep the same functionality as before, but faster :)

#22 - 06/27/2023 09:56 AM - Eric Faulhaber

The purpose of confirmUniqueConstraintViolation: consider that we are validating record A, which is in the FWD server's memory, but not yet flushed. We do the unique query check for record A, and we find that there is a record (call it B') in the database, whose values on a particular unique index are the same as those of record A.

This would cause a unique constraint violation if we flushed record A at this moment, *unless* record B' in the database also was represented by a changed, as-yet-unvalidated and unflushed DMO in the FWD server's memory (call that record B), with one or more of those unique index values modified, such that they are not the same as those of record A.

In this situation, the potential unique constraint violation we detected on record A is "cured" by the FWD server's in-memory state of record B. The idea is that the changed state of record B' (i.e., that represented by record B in memory) will be flushed in the same transaction as record A, and the combination of these flushes will avoid the detected unique constraint violation. Note that the changed record B is still subject to its own validation; the current action is all about making sure record A is unique, given the combined database and FWD in-memory state.

I'm not sure how this fits into your changes, but I suspect something like this may still be necessary, if we are left with modified state being tracked in both the database and FWD server at the same time, for multiple records.

#23 - 06/28/2023 10:46 AM - Alexandru Lungu

> The purpose of confirmUniqueConstraintViolation: consider that we are validating record A, which is in the FWD server's memory, but not yet flushed. We do the unique query check for record A, and we find that there is a record (call it B') in the database, whose values on a particular unique index are the same as those of record A.

Well, this means that if we do INSERT INTO [...] VALIDATE and fail, we may need to do a confirmUniqueConstraintViolation. If this cures the violation, then we should go ahead with a second INSERT INTO [...] (without validate). This basically means 2 INSERT attempts vs. one UNION SELECT and one INSERT attempt (note that I omitted the SELECT done by confirmUniqueConstraintViolation from the calculation).

I really wonder now how often this case actually occurs. Can we compromise two INSERT for such case just to allow one single INSERT in the other cases (from my POV, more common)? The downside is that an INSERT updates all indexes, while our former SELECT UNION was targeting only invalidated unique indexes.

I think we will need a statistic here. Radu, please feel free to count how many such cases actually occur in a customer application. Please consider creating a JMX and commit it to 7323a. There should be some kind of INSERT VALIDATE SUCCESS kind of JMX and INSERT VALIDATE FAILED kind of JMX. Please consider only VALIDATE attempts.

#24 - 06/28/2023 11:04 AM - Eric Faulhaber

Alexandru Lungu wrote:

> I really wonder now how often this case actually occurs.

Yes, without some statistics, I cannot say how often a "cure" actually occurs in real use. It seems in production code, a unique constraint violation should be pretty rare (compared to validations that are successful), and this cure action even more rare.

I can say that I added this check because it was actually happening in real code, and I had to resolve a false negative validation at some point. Unfortunately, I cannot remember the use case, nor whether it was ever actually written up as a ticket.

#25 - 06/29/2023 03:14 AM - Alexandru Lungu

> I think we will need a statistic here. Radu, please feel free to count how many such cases actually occur in a customer application. Please consider creating a JMX and commit it to 7323a. There should be some kind of INSERT VALIDATE SUCCESS kind of JMX and INSERT VALIDATE FAILED kind of JMX. Please consider only VALIDATE attempts.

Once ready, please post a patch with the JMX here. I will like to do some tests with it myself afterward.

#26 - 07/04/2023 04:23 AM - Radu Apetrii

I've committed to 7323a, rev. 14591 the following changes:

Alexandru Lungu wrote:

> Radu, consider the right changes to transform validateMode into a allowDBUniqueCheck.

I've changed the name of validateMode into allowDBUniqueCheck.

> Databases that can't suppress unique checks will log a SEVERE, but still do the insert/update.

I've allowed them to still do the insert/update, but I've decided to not log the cases in which the database can't do the unique check itself. The reason behind this is that the server log might become overpopulated with these messages, taking into account how many statements are executed against a persistent database. Also, I believe that as long as the insert/update is executed, the log message isn't very relevant. Please correct me on this one if I'm wrong.

- My only concern is related to validatedMode ?. I think you should do a warning logging and fallback to insertSql[i] if validateMode is true and insertValidateSql[i] is null. Otherwise, a misconfiguration here (e.g. validateMode on true for a persistent database) may lead to NPE, which is quite fatal.

I've done something similar to this suggestion. Mainly, I've allowed the execution of a statement with VALIDATE only if the database can validate the index itself and there is a SQL with the keyword in it. This means that when allowDBUniqueCheck is false, or the insertValidateSql[i] is null, then the insert/update will be executed without VALIDATE.

I will also post a patch with the JMX very soon.

#27 - 07/04/2023 05:09 AM - Radu Apetrii

- File 7323-JMX.patch added

Here's a patch for measuring how many inserts/updates are executed in total/successfully with VALIDATE and seeing how many of them were cured by confirmUniqueConstraintViolation. This can be applied to 7323a, rev. 14591.

Alexandru Lungu wrote:

> There should be some kind of INSERT VALIDATE SUCCESS kind of JMX and INSERT VALIDATE FAILED kind of JMX. Please consider only VALIDATE attempts.

I've created an INSERT_VALIDATE_TOTAL for counting how many inserts/updates with VALIDATE are there and an INSERT_VALIDATE_SUCCESS for counting how many of them were successful. Naturally, their difference will give you the number of failed inserts/updates. Also, from the failed ones, you can see how many inserts were cured, by setting the INSERT_VALIDATE_CURED.

Alexandru Lungu wrote:

> Well, this means that if we do INSERT INTO [...] VALIDATE and fail, we may need to do a confirmUniqueConstraintViolation.

This does happen. I believe the part with the second insert (without VALIDATE) needs to be added.

In terms of statistics, with the customer application that I tested, the succes rate of insert/updates with VALIDATE was **100%**, which should be really promising. I will wait for your round of testing before I get too excited.

#28 - 07/04/2023 05:20 AM - Alexandru Lungu

> I've allowed them to still do the insert/update, but I've decided to not log the cases in which the database can't do the unique check itself. The reason behind this is that the server log might become overpopulated with these messages, taking into account how many statements are executed against a persistent database. Also, I believe that as long as the insert/update is executed, the log message isn't very relevant. Please correct me on this one if I'm wrong.

This might cause a functional mismatch. If we send allowDBUniqueCheck with false, but the database isn't able to suppress unique check, then an unexpected ValidationException is thrown. At that point, the server reaches an invalid state, and we might want to have a feedback on that - a LOG. I see you point, but the alternative from my POV is to throw an InvalidStateException, which may be way worse. We can leave it as it is now, as the ValidationException itself is quite descriptive, but please use this logging in your tests just to check that it isn't ever logged.

> I've created an INSERT_VALIDATE_TOTAL for counting how many inserts/updates with VALIDATE are there and an

> INSERT_VALIDATE_SUCCESS for counting how many of them were successful. Naturally, their difference will give you the number of failed inserts/updates. Also, from the failed ones, you can see how many inserts were cured, by setting the INSERT_VALIDATE_CURED.

Cool! I am waiting for the regression fix now, so at that point I am able to do the proper tracking (#7323-21). If I am doing the JMX straight-away, the validation will break with first very soon. Radu, please focus on getting that cure for validate mode.

#29 - 07/04/2023 07:24 AM - Radu Apetrii

- File 7323-JMX.patch added

Alexandru Lungu wrote:

> Cool! I am waiting for the regression fix now, so at that point I am able to do the proper tracking (#7323-21). If I am doing the JMX straight-away, the validation will break with first very soon. Radu, please focus on getting that cure for validate mode.

I've committed to 7323a, rev. 14592. Basically, if a flush with VALIDATE fails, but it gets cured, now a second flush takes place, but without having VALIDATE.

Also, I attached the updated patch with JMXs since it needed a slight change.

#30 - 07/17/2023 07:57 AM - Alexandru Lungu

Radu, I tested 7323a and I still have Unique index or primary key violation (lots). I've debugged a bit and I've seen that the cure is attempted but without success. It finds the cached records, but it doesn't have the CHANGED flag, so it is not considered. The test I have is quite huge and I can't extract it separately. Please feel free to take a pause on your current task and take a look on this matter. Thank you!

I will do my best to attempt extracting a test-case for you, but I can't guarantee I can get it easily.

#31 - 07/18/2023 04:52 AM - Radu Apetrii

Alexandru Lungu wrote:

> I will do my best to attempt extracting a test-case for you, but I can't guarantee I can get it easily.

I will try to find a smaller example in the meantime. I will keep you up to date regarding this.

#32 - 07/18/2023 09:03 AM - Alexandru Lungu

Radu, this is something that triggered the error, but note that the code is only a cut-down, so you will need to set it up properly:

```
create data-source ds.

create buffer b1 for table "t1".
create data-source ds1.
ds1:add-source-buffer(b1, "id_t1").

create buffer b2 for table "t2".
create data-source ds2.
ds2:add-source-buffer(b2, "id_t2").

ds1:save-where-string("t1") = "where t1.id_t1 = 1".
ds2:save-where-string("t2") = "where t2.id_t2 = 2".

ds:get-buffer-handle("tt1"):attach-data-source(ds1).
ds:get-buffer-handle("tt2"):attach-data-source(ds2).

ds:get-buffer-handle(1):data-source:fill-where-string = "...". // some where clause
ds:fill().
```

This code was not in an explicit do transaction block, but simply in an OO method. However, note that fullTx is true and there is no savepoint created when the error occured.

#33 - 07/19/2023 07:07 AM - Radu Apetrii

I've analyzed the problem a bit and it seems that those Unique index or primary key violation appear because the first attempt of INSERT/UPDATE fails (the one with VALIDATE), then it gets logged and only after that the cure process is tried.

In the original scenario (before the changes from 7323a), the logging part in between did not exist, there was only one logging happening after the cure process. After the changes, two "loggings" were happening: one placed between the INSERT/UPDATE and the cure, and one after the cure was finalized. Thus, I've deactivated the first logging in case a UNIQUE VIOLATION occurs, in order to match the original behavior.

The changes are on 7323a, rev. 14593.
Note that the execution of the program was not affected in any way, the problem was just that some **unnecessary logging was happening**.

Alex, when you have the time, can you do another run and confirm that these messages no longer appear? Thank you!

#34 - 07/20/2023 04:42 AM - Radu Apetrii

I've refactored the logging process when handling a SQLException in an INSERT/UPDATE. Instead of only checking the given exception, the program now goes through the exception chain and logs all the causes that are not UNIQUE VIOLATION. The commit is on 7323a, rev. 14595.

#35 - 07/20/2023 05:54 AM - Alexandru Lungu

Tested with 7323a/rev. 14595 with a simple POC application and got no regression. I am planning to use the JMX you provided in #7323-29. Please feel free to add the cases where the validation fails completely, even after cure. Also, please make a JMX over 7323a (without any validate changes) to check the same things. I want to compare them so:

- number of validations that fail is the same
- the number of cures is the same
- the number of failing cures is the same

Thank you!

#36 - 07/20/2023 06:54 AM - Alexandru Lungu

Good news: this also passes the full-regression tests I use. The only concern now is the time - I will start a profiling round once the JMX is ready (so I can set an expectancy on time improvement before-hand).

#37 - 07/20/2023 10:01 AM - Radu Apetrii

- File 7323-JMX-before.patch added

Alexandru Lungu wrote:

> Also, please make a JMX over 7323a (without any validate changes) to check the same things.

I've attached such a patch. By applying it, one will be reverted to 7323a, rev. 14586 (the revision from before the changes) and some JMXs will be added. The things that can be measured are:

- INSERT_UPDATE_TOTAL, the total number of insert/update statements that are validated and flushed. This is equivalent to INSERT_UPDATE_VALIDATE_TOTAL from #7323-29.
- INSERT_UPDATE_SUCCESS, the number of insert/update statements that are validated and flushed that were executed successfully. The equivalent is INSERT_UPDATE_VALIDATE_SUCCESS.
- INSERT_UPDATE_CURED, the number of insert/update statements that failed the validation, but managed to be cured. The equivalent is INSERT_VALIDATE_CURED.

To obtain:

- The number of validations that fail, one can subtract SUCCESS from the TOTAL.
- The number of cures, one can check CURED.
- The number of failing cures, one can subtract SUCCESS and CURED from the TOTAL.

Let me know if there is something else to do.

#38 - 07/21/2023 11:13 AM - Alexandru Lungu

I tested with the JMX and everything is fine. The number of inserts / cures and failed cures is the same, good job.
However, the profiling tests aren't encouraging. The total improvement is only -0.3%. I doubt that this whole process is not improving by more, so I will double check the solution. I am sure we are missing something here.

EDIT: Radu, please double-check the implementation and report anything that you may see fit for improvement. You can also make a dummy test to do your profiling and compare with the previous version. Maybe we can work with FWD-H2 for this.

#39 - 07/24/2023 04:55 AM - Alexandru Lungu

I think the numbers are quite relevant, so I will write them here:

JMX	Before	After
Total	29.912	29.226
Success	29.540	28.854
Fail Cure	372	372
Cured	0	0

There are slight differences, but these are due to a non-deterministic test-case I run. Anyway, the number of cures (success / fail) is the same, so I tend to say the changes are right. My whole point here is that 99% inserts are success with no cure. Therefore, in 99% cases, the removal of the validation SELECT (with UNION) was worth it. Clearly we should see better improvement as now we have ~28k less SELECT statements, right?

#40 - 08/03/2023 04:28 AM - Radu Apetrii

Alexandru Lungu wrote:

> Clearly we should see better improvement as now we have ~28k less SELECT statements, right?

I think the problem might not be that VALIDATE is not fast enough, but the fact that it appears not as often as we would have expected. Thus, its performance improvement gets lost in the overall time of the program. Let me break this down into two cases:

1. An example of just VALIDATE queries
I tried a test with 1 million INSERT statements and nothing else. Without the changes, they would be executed normally, while with the changes, all of them would contain VALIDATE.

- Without the changes, the whole program runs in ~6 seconds.
- With the changes, the program runs in ~4 seconds. This improvement is definitely noticeable, standing at 33%.

2. An example based on the queries from a customer application
I extracted the statements executed over the _temp database from a customer application and noticed that only ~16.66% of the total INSERT + UPDATE statements contain VALIDATE. Not only that, but only 1 out of 10 VALIDATE statements is an UPDATE. So I created a test keeping in mind the percentages that appeared in the extracted statements file.

	After the changes	Before the changes
VALIDATE count	30,000	0
SELECT count	100,000	130,000

INSERT count	27,000 (with VALIDATE) out of 80,000 in total	80,000
UPDATE count	3,000 (with VALIDATE) out of 100,000 in total	100,000
Average time	4979 ms	5094 ms

Even though there are ~30k less SELECT statements, the improvement is very thin.
Please correct me if I misunderstood your question.

#41 - 08/03/2023 05:00 AM - Alexandru Lungu

I don't expect a miracle from VALIDATE, but 30k SELECT statements are in fact a lot (from my POV). I agree that they are usually the same and are already prepared and cached. But even so, according to your results, there is a ~2.2% improvement. This is a relevant improvement comparing to my 0.3%. Of course, I don't expect -10% improvement, but still I am trying to push my tests over (or in this case - under) the -2% line.

Also, my test has ~30k inserts/updated, just like yours, so we are on par here. I will retry the test, but I want to be a final one. For that matter, I want "the best optimizations" to be in place at that point. That is, I expect the most efficient code from that point of view, including the FWD-H2 code. Note that for #7323-17 we missed pre-computing the VALIDATE statement, which implied tons of string concatenations in the end. I wonder if we are still missing things like his now (either in FWD or FWD-H2):

- is the cure really time consuming that it actually slows-down the process?
- is the save-point creation an issue here?
- is the psCache properly doing the caching?

Now that we do both un-validated inserts/updates **and** validated inserts/updates, we need to prepare and cache 2 insert/update statements per DMO. This may be quite bad for short-lived dynamic DMOs. Maybe we can so some kind of direct-access "enable/disable validation*, so that we get rid of the trailing VALIDATE and its preparing + string concatenation **Radu, what is your input on this idea?**

```
// direct-access startValidate()
INSERT INTO [...]; // no more trailing VALIDATE, keep only SOFT-UNIQUE keyword
// direct-access stopValidate()
```

#42 - 08/03/2023 05:05 AM - Radu Apetrii

Alexandru Lungu wrote:

> **Radu, what is your input on this idea?**

I will investigate the aspects mentioned and I will get back with updates shortly.

#43 - 08/03/2023 08:05 AM - Radu Apetrii

I've committed to 7323a, rev. 14690 and 7323a_h2, rev. 23. I added support for the DirectAccess idea mentioned in #7323-41, meaning that the SQL no longer contains the keyword VALIDATE. Instead, H2 gets notified by DirectAccess that it should handle unique index validation.

Performance wise, the time of the second test from #7323-40 decreased from 4979 ms to a very promising 4481 ms.
I will wait for your opinion upon these changes. Until then, I will take another look, maybe we missed something else.

#44 - 08/08/2023 04:40 AM - Alexandru Lungu

- % Done changed from 70 to 90

I retested with the latest changes in 7323a and got -1% improvement now. This starts looking good. At this point I am OK with the solution performance-wise.

I will do a set of regression testing and a final review. If you have anything left here, please let me know. Anyway, this won't be merged until we fix FWD-H2 rev. 1.24 (causing an infinite loop on a customer application).

#45 - 08/21/2023 07:13 AM - Alexandru Lungu

- % Done changed from 90 to 100

- Status changed from WIP to Review

Radu, I am planning to merge **7323a_h2** to FWD-H2. Please double check that everything is alright.

Please focus on identifying any changes that may cause regressions when VALIDATE mode is not used. The point is that the FWD solution will be merged later on, so we need a sound solution after upgrading FWD-H2 alone.

#46 - 08/24/2023 03:18 AM - Radu Apetrii

Alexandru Lungu wrote:

> Please focus on identifying any changes that may cause regressions when VALIDATE mode is not used.

I've done some tests with the most recent version of 7323a_h2 (the one intended to be merged into FWD-H2) and the version from before the changes of 7323a (rev. 14680). There were no errors/regressions found in neither adaptive_scrolling tests, testFWD tests, nor a large customer application. This means that the merge process has a **green light** from my point of view.

#47 - 08/24/2023 08:27 AM - Alexandru Lungu

Committed 7323a_h2 as FWD-H2 rev. 26 and archived.

The FWD changes will be merged once we upgrade to the latest FWD-H2.

#48 - 10/25/2023 03:53 AM - Alexandru Lungu

Rebased 7323a to latest trunk. 7323a is now at rev. 14799.

#49 - 10/25/2023 07:00 AM - Alexandru Lungu

- % Done changed from 100 to 70

- Priority changed from Normal to High

- Status changed from Review to WIP

I attempted to profile 7323a and succeeded into actually running it, but the time is way off (+3-4%). After some review, I actually have some concerns on the current state of the implementation:

- minor: there are some files that have only history entries (no changes). Please remove the history entries entirely
- major: don't cache validateDriver. There is one driver per session. If you have one single driver caches, you leak information between users. Thus, one can start/stop validation of another user. This should be fixed asap - just remove the caching.
- major: I don't think it is enough to enable/disable validation when preparing the statements. Note that the same statement can be used when validating / not-validation. In your implementation, the validation state will incorrectly "stick" to the prepared statement for its entire lifecycle. **We should enable/disable it only when executing**! Radu, consider moving away from the VALIDATE keyword entirely.

The problems are quite major, so I will stop testing. Radu, please queue this task and mark it with higher priority.

#50 - 10/25/2023 07:01 AM - Alexandru Lungu

Also:

- major: mind moving stopValidating in a finally clause. Otherwise, an exception will miss the stopping of the validation process. The state of validating will leak, so that next update attempt will validate automatically.

#51 - 11/01/2023 06:19 AM - Radu Apetrii

Alexandru Lungu wrote:

> Radu, please queue this task and mark it with higher priority.

On it. I also noticed some date mismatches in a few history entries, so I will fix them too.

#52 - 11/06/2023 02:19 AM - Radu Apetrii

- Status changed from WIP to Review

- % Done changed from 70 to 100

I've committed to 7323a rev. 14800 the following changes:

- Removed the validateDriver caching.
- Moved startValidate and stopValidate around the query execution instead of preparation.

- Placed stopValidate in a finally block.
- Removed redundant/unnecessary history entries and fixed some dates.

#53 - 11/13/2023 11:22 AM - Alexandru Lungu

- Status changed from Review to Internal Test

Review of 7323a

- I don't think the cast to P2JH2Dialect is needed anymore in TempTableHelper. The methods are already in Dialect
- In Persister, DirectAccessHelper.hasDirectAccess(session) was already computed in a separate variable.

These are minor changes, so I can go ahead with Internal Testing.

#54 - 11/14/2023 02:18 AM - Radu Apetrii

Is it OK to commit those changes now or will it interfere with your testing?

#55 - 11/14/2023 03:39 AM - Alexandru Lungu

I will start testing in ~30mins. If you have time to get them committed, it is ideal. If not, the changes are minor anyway, so no problem to delay.

#56 - 11/14/2023 04:07 AM - Radu Apetrii

Alexandru Lungu wrote:

> I will start testing in ~30mins. If you have time to get them committed, it is ideal. If not, the changes are minor anyway, so no problem to delay.

OK. 3 minutes should be enough time to allow me to commit the changes.

#57 - 11/14/2023 04:39 AM - Radu Apetrii

I committed to 7323a, rev. 14801 the following changes:

- Reverted TempTableHelper to its original state.
- Replaced that DirectAccessHelper.hasDirectAccess(session) call with the variable name.
- Replaced PersistenceException with RuntimeException in Dialect.getCreateIndexString.

#58 - 11/14/2023 06:45 AM - Alexandru Lungu

Unfortunately, I still have the same results as in #7323-49 :/ We are adding to much overhead somewhere. Maybe after the rabase / solution fixing, we added some complexity that is unwanted. I am really giving faith in this approach, but we need to clarify what we added that is actually slowing down the process. I will do some tests in the mean-time.

#59 - 11/14/2023 08:02 AM - Alexandru Lungu

- Priority changed from High to Urgent

I have found only some points:

- Remove hasDirectAccess entirely and let startValidate and stopValidate simply return if there is not direct-access connection (making them no-op).
- Move startValidate and stopValidate at the very beginning of the methods to also catch batch operations **or** to avoid doing multiple startValidate for tables with normalized extents.
- In allowDBUniqueCheck, move temporary to be the first parameter.
- Make allowDBUniqueCheck false if the check bit-set is empty (mind the short-circuit from validateUniqueByQuery
- Cache the whatever you can in Persister class - it is one per session (for instance, mind computing DirectAccessDriver only once in Persister an avoid the DirectAccessHelper.
- Maybe you can avoid passing allowDBUniqueCheck as parameter and detect it at Persister level (only once).

Radu, please focus on this task asap. Consider making some performance tests yourself. We need a conclusion on this one: **go** or **no go**.

#60 - 11/22/2023 05:13 AM - Alexandru Lungu

Also, please attempt to retest the POC for performance after the changes. Make a baseline with 7156b / an intermediary profile with 7323a (without #7323-59 fixes) and a final profile with 7323a and fixed. Mind debugging the application if you feel like you need more insight onto onto how the solution is running.

#61 - 11/27/2023 10:04 AM - Radu Apetrii

I've addressed the review points and committed to 7323a rev. 14802-14803. Right now I'm running the performance tests, but so far I'm seeing just a very slim improvement on the default 7156b. I'll create a report once they are done.

Also, I'll keep checking if things can be improved further, but I want to raise a question: **Is it possible that the changes in FWD-H2 also contribute to the total overhead?** Maybe things can be done more efficiently there too.

#62 - 11/27/2023 10:14 AM - Alexandru Lungu

Theoretically yes. Practically, the flag will only take affect if there are violations. We need to check on our large POC if there are several such violations or not.
Mind that in order to report such violation, we need to throw an exception which is quite time consuming. Please check.

#63 - 11/28/2023 06:38 AM - Radu Apetrii

I finished running the performance tests and the results are not very encouraging:

- 7323a rev. 14801 (**before** the changes from #7323-59) is ~3.8% **slower** than the default 7156b.
- 7323a rev. 14803 (**after** the changes from #7323-59) is ~0.02% **faster** that the default 7156b.

Out of curiosity, I noticed that there are 0 violations out of 243k inserts + updates on a warmup + one test on POC performance tests.

#64 - 11/28/2023 06:46 AM - Alexandru Lungu

Radu, can you check if we are still hitting validateUniqueByQuery for _temp? If so, how many times do we actually bypass validateUniqueByQuery using allowDBUniqueCheck vs how many times is validateUniqueByQuery actually called. Test this against a warm run on POC.

#65 - 11/28/2023 08:02 AM - Radu Apetrii

Alexandru Lungu wrote:

> Radu, can you check if we are still hitting validateUniqueByQuery for _temp? If so, how many times do we actually bypass validateUniqueByQuery using allowDBUniqueCheck vs how many times is validateUniqueByQuery actually called. Test this against a warm run on POC.

I'll check that. Before reviewing the changes, allow me to modify one more thing as I noticed a slight performance difference.

#66 - 12/06/2023 05:08 AM - Radu Apetrii

I've committed to 7323a rev. 14804-14805 some minor changes regarding:

- Fixed the validation condition in Validation.validateUniqueIndices.
- Moved the computation of allowDBUniqueCheck in the constructor of Persistence.
- Removed rev. 14803 changes as they were no longer needed once rev. 14804 had been added.

Still, testing shows a decrease in performance, meaning that this is not faster than the normal 7156b. Thus, I've expanded the statistics done with the following observations (all on warm runs):

- There are a lot of skipped validations when using the changes from 7323a. ShowHide
  - With changes, out of **33478** function calls (Validation.validateUniqueIndices), **33311** validations (Validation.validateUniqueByQuery) are skipped due to allowDBUniqueCheck, **0** due to the check emptiness condition, **0** due to (multiplex != null || !flush), and **167** validations are executed.
  - Without changes, out of **31832** function calls, **0** validations are skipped, and **31832** validations are executed.

- In the tests that I'm running, check always contains something. ShowHide
  - With changes, all **33478** calls (Validation.validateUniqueIndices) contain a check that is neither null, nor empty.
  - Without changes, all **31832** calls contain a check that is neither null, nor empty.

- There are more function calls (Validation.validateUniqueIndices) when using the changes. This I find to be quite strange and the only explanation I have for it is that some queries fail to execute when using startValidate-stopValidate, so they require a second execution without it. ShowHide
  - With changes, on a total of 4 runs (+ warmup), there are **108478** function calls.
  - Without changes, on a total of 4 runs (+ warmup), there are **104648** function calls.
  - This means that on average, on a 100 runs test, there will be ~**12k** more function calls when using the changes from 7323a. Even though there are lots of calls added, I don't think it is enough to motivate the slowdown in performance, especially considering the fact that most of the calls skip the validation (Validation.validateUniqueByQuery) process.

I recall checking this before, but I would like to re-create some statistics about how many queries fail the startValidate-stopValidate process, so that they will be executed again. I'll come back with results.

Update: There are **0** startValidate-stopValidate processes that fail. I'm quite confused now.

#67 - 03/13/2024 04:56 AM - Constantin Asofiei

I'm rebasing 7323a and 7323a_h2 to check them with #8363.

#68 - 03/13/2024 05:10 AM - Constantin Asofiei

Constantin Asofiei wrote:

> I'm rebasing 7323a and 7323a_h2 to check them with #8363.

7323a_h2 was already merged to fwd-h2 trunk.

Radu: rebasing 7323a has a lot of conflicts - can you take care of this, please?

#69 - 03/13/2024 05:14 AM - Radu Apetrii

OK, I'll handle this now since I'm waiting for an import anyways.

#70 - 03/13/2024 06:34 AM - Radu Apetrii

Done. Rebased 7323a with trunk rev.15051, meaning that 7323a is now at rev. 15066.

#71 - 03/13/2024 06:36 AM - Constantin Asofiei

Radu Apetrii wrote:

> Done. Rebased 7323a with trunk rev.15051, meaning that 7323a is now at rev. 15066.

Please check, there are compile errors.

#72 - 03/13/2024 06:42 AM - Radu Apetrii

Constantin Asofiei wrote:

> Radu Apetrii wrote:
>
> > Done. Rebased 7323a with trunk rev.15051, meaning that 7323a is now at rev. 15066.
>
> Please check, there are compile errors.

Are you sure that you are running rev. 15066 and not 15065? I did do a commit a few minutes ago to fix a duplicate code issue after the rebase. I even did a gradlew all and ran an example to make sure that it works fine.

#73 - 03/13/2024 06:45 AM - Constantin Asofiei

Radu Apetrii wrote:

> Constantin Asofiei wrote:
>
> > Radu Apetrii wrote:
> >
> > > Done. Rebased 7323a with trunk rev.15051, meaning that 7323a is now at rev. 15066.
> >
> > Please check, there are compile errors.
>
> Are you sure that you are running rev. 15066 and not 15065? I did do a commit a few minutes ago to fix a duplicate code issue after the rebase. I even did a gradlew all and ran an example to make sure that it works fine.

The branch was unbound completely, forgot to clean it after the attempted rebase. Thanks.

#74 - 03/13/2024 12:43 PM - Constantin Asofiei

I've tested with #8363 and there is no impact or makes things slightly worse in some cases.

#75 - 03/14/2024 10:46 AM - Constantin Asofiei

Alexandru/Radu, I think we may need a call to discuss the soft index approach.

Radu: I took a look again at 7323a after rebase and some files have only header changes (compared with 15051). Is this expected?

#77 - 03/15/2024 12:30 PM - Constantin Asofiei

I've discussed this morning with Radu what this task is meant to do and I think I understand it: let H2 throw an exception at flush, in other words, when 'validate mode' is enabled, and FWD will use this as unique constraint validation checks. This makes sense in theory, as Alexandru mentioned, 'we don't want to walk the indexes twice, once for the unique validation and the other for actual insert'.

But, I think I found some issues with the soft-unique support:

- in Validation.validateUniqueIndices:

```
         if ((multiplex != null || !flush))
         {
            // validate by executing unique index queries
```

```
            validateUniqueByQuery(check);
         }

         if (flush || buffer.isAutoCommit())
         {
            Persister persister = session.getPersister();
            if (persister.getAllowDBUniqueCheck() == null)
            {
               // allowDBUniqueCheck can be set to true only when dealing with temporary databases
               // allowDBUniqueCheck is used to determine if H2 will do index validation or should this
               // be done normally (server-side validation)
               boolean isCheckEmpty = check != null && check.isEmpty();
               boolean allowDBUniqueCheck = !isCheckEmpty && canSkipServerSideUniqueCheck();
               persister.setAllowDBUniqueCheck(allowDBUniqueCheck);
            }

            // validation will be done by the database as part of the flush (for persistent tables)
            flush();
         }
```

validateUniqueByQuery is still called for temp-tables. Why is that?

- the soft-unique index is still added the recid trailing column - this is incorrect, as these will always be set before the record is flushed, by FWD, and they will be unique. Did this meant to have the soft-unique indexes as a 'duplicate' for the original index, but without the trailing 'recid' column?

#78 - 03/15/2024 01:06 PM - Radu Apetrii

Constantin Asofiei wrote:

> validateUniqueByQuery is still called for temp-tables. Why is that?

This shouldn't be the case, as the main reason was to avoid the validateUniqueByQuery call for temp-tables plus flush cases. Let me double-check the code after the rebase to make sure that things weren't lost in the process. I'll get back with an update soon.

#79 - 03/15/2024 03:23 PM - Constantin Asofiei

I've placed on devsrv01:/tmp/7323a_rev_14805_trunk_14789.7z the pre-rebase branch, in case you need it.

#80 - 03/15/2024 08:47 PM - Radu Apetrii

I committed to 7323a rev. 15067 a rebase fix for classes Persistence, Persister, and Validation. It seemed that after the rebase, the main point of these changes got lost (avoiding the extra index validation).

Constantin, when you have some time, please have another go with 7323a.

#81 - 03/16/2024 07:11 AM - Constantin Asofiei

What we need is to mark the recid trailing component of a 'soft-unique' index as soft-unique, too - so this can be skipped in BaseIndex.compareRows, when validateMode is enabled. The alternative would be to leave the main index as is (not soft-unique) and create an additional index which is soft-unique, but this adds overhead of maintaining two indexes.

Radu, can this be done easily in H2 (add soft-unique option at a index column)? This would mean that in FWD we will need to mark the recid field in a soft-unique index as soft-unique, for temp-tables.

In the current build, soft-unique doesn't work at all as collisions can't be found because of the recid trailing column added to the 'soft-unique' temp-table indexes (btw, we can't remove the trailing recid).

#82 - 03/17/2024 12:22 PM - Constantin Asofiei

Created task branch 7323b_h2 from fwd-h2 trunk rev 40.

In rev 41, it adds a "soft_unique" option for index columns.

In 7323a rev 15068, it adds 'soft_unique' at the temp-table recid column for unique indexes.

Radu/Alexandru: can you do a review? Is there anything else to do in 7323b_h2? My question is if there can be anything wrong that H2 caches the Insert/Update commands and this flag remains set.

Alexandru: can you regression test this with the POC/7156b app?

We will need to push H2 trunk rev 41 to xfer once 7323b_h2 is released.

As a side note: with #8363, I get a ~8% overall improvement. I'm regression testing this with ETF.

#83 - 03/17/2024 03:04 PM - Constantin Asofiei

Constantin Asofiei wrote:

> I'm regression testing this with ETF.

ETF testing passed.

As a side note: we may want to rename the SOFT_UNIQUE option at the index column to something else.

Eric: you may want to review 7323a, too.

#84 - 03/19/2024 12:36 AM - Eric Faulhaber

Code review 7323a/15068:

The changes look ok to me, though it is a little confusing that the SOFT_INDEX keyword is used to modify both an index and a column of an index.

I also reviewed 7323a/15052-15067, mostly for context. Session has some orphaned header file entries, but no corresponding changes in the code. This seems like a rebase problem, but I don't know if the problem is with the header or the code.

I also looked over 7323b_h2/41 and briefly at 7323a_h2/26-28 (to get a better understanding of the core set of SOFT_UNIQUE changes). While the 7323b_h2 changes looked ok to me, I am not very familiar with the H2 internals, so I will rely on Alexandru and Radu to manage the changes there.

I do think in general we should be more verbose with our comments in the H2 code (despite the fact that the original code is poorly commented), especially since we are making changes which depart from standard SQL behavior and are bespoke to FWD.

If Radu/Alexandru are ok with the H2 changes, we should move ahead.

#85 - 03/19/2024 04:18 AM - Alexandru Lungu

Review of 7323b_h2

- I reviewed Radu's changes in the past and they were OK.
- Recent changes are fine to me. Indeed, it is quite confusing that we have soft_index for both the index and the column even in H2. But as long as this is only for _temp H2, I guess this is just an "internal" issue that may affect the debugging.
  - I was thinking for a while at the changes to ignore the recid in that way (simply continue when doing a row to row comparison). I concluded that it is OK and doesn't affect the behavior of H2 in other contexts. I was thinking more precisely at deletes or updates and how they are going to traverse the TreeIndex to find that records - but in that case they are not in a "validate" mode, so they will continue working as usual. It looks fragile to me (relying only on a "validate" flag), but it works! No other suggestion from my side.
- Long story short: I am OK with the changes.

#86 - 03/19/2024 09:27 AM - Greg Shah

- Status changed from Internal Test to Merge Pending

You can merge now.

#87 - 03/19/2024 09:52 AM - Alexandru Lungu

Constantin, you need to merge H2 first, build, upload the FWD binary, commit to the FWD branch the upgrade and **after** merge the FWD branch.

#88 - 03/19/2024 10:11 AM - Constantin Asofiei

Yes, working on it.

#89 - 03/19/2024 10:31 AM - Constantin Asofiei

- Status changed from Merge Pending to Test

Branch 7323a was merged into trunk as rev. 15065 and archived.

7323b_h2 was merged into FWD-H2 trunk rev 41 and archived.

#90 - 03/19/2024 01:52 PM - Constantin Asofiei

Found a regression in 7323a changes:

```
--- src/com/goldencode/p2j/persist/orm/Validation.java
+++ src/com/goldencode/p2j/persist/orm/Validation.java
@@ -484,7 +484,9 @@
             allowDBUniqueCheck = canSkipServerSideUniqueCheck();
          }

-         if ((multiplex != null || !flush) && !isCheckEmpty && !allowDBUniqueCheck)
+         if ((multiplex != null || !flush) &&
+             !isCheckEmpty                 &&
+             (!allowDBUniqueCheck || (buffer.isTemporary() && !buffer.isAutoCommit())))
          {
             // validate by executing unique index queries
             validateUniqueByQuery(check);
```

The scenario is:

- there is a TRANSACTION open
- the field is updated via ASSIGN ... NO-ERROR.
- at the end of the batch assign, there must be validation, but not flush, and ERROR-STATUS:ERROR flag to be set.
  - with 7323a changes, the if (flush || buffer.isAutoCommit()) condition prevents the flush, and the if ((multiplex != null || !flush) && !isCheckEmpty && !allowDBUniqueCheck) condition prevents the unique validation - this is done too late.
  - buffer.isAutoCommit() is false for temp-tables while in a TRANSACTION.

Please advise.

#91 - 03/20/2024 02:58 AM - Constantin Asofiei

- Status changed from Test to Review

Created task 7323b from trunk rev 15073. The fix in previous note is in rev 15074. Please review.

#92 - 03/20/2024 04:40 AM - Alexandru Lungu

AFAIK, the condition there was rewritten to consider DBUniqueCheck, but remain equivalent to what was before. I prefer to study how the condition

was before the changes, if it is not equivalent and the reason is the on in #7323-90.

Radu/Constantin: can you share how the condition looked like before the 7323a changes?

#93 - 03/20/2024 04:48 AM - Radu Apetrii

Alexandru Lungu wrote:

> Radu/Constantin: can you share how the condition looked like before the 7323a changes?

This is how the code looked like before the 7323a changes:

```
         if (multiplex != null || !flush)
         {
            // validate by executing unique index queries
            validateUniqueByQuery(check);
         }

         if (flush || buffer.isAutoCommit())
         {
            // validation will be done by the database as part of the flush (for persistent tables)
            flush();
         }
```

#94 - 03/20/2024 05:30 AM - Alexandru Lungu

Then I agree with Constantin's changes. Before, the 7323a changes, the flush (so the DB validation0 was done only if flush || buffer.isAutoCommit() holds. Basically, allowDBUniqueCheck should be true only if the dialect allows **and** (we are going to flush **or** the buffer is auto-commit). It seems like the second part was lost in the mean-time. We need to ensure that validation should still happen when flush is false **or** the buffer is not auto-commit. I would rather change to:

```
=== modified file 'src/com/goldencode/p2j/persist/orm/Validation.java'
--- old/src/com/goldencode/p2j/persist/orm/Validation.java    2024-03-16 00:42:23 +0000
+++ new/src/com/goldencode/p2j/persist/orm/Validation.java    2024-03-20 09:31:14 +0000
@@ -481,7 +481,7 @@
          }
          else
          {
-            allowDBUniqueCheck = canSkipServerSideUniqueCheck();
+            allowDBUniqueCheck = canSkipServerSideUniqueCheck() && (flush || buffer.isAutoCommit());
          }

          if ((multiplex != null || !flush) && !isCheckEmpty && !allowDBUniqueCheck)
```

This covers Constantin's regression **and** the case when we don't want to flush the record, but still want to validate. allowDBUniqueCheck includes the condition that should hold for the flush (flush || buffer.isAutoCommit).

Please review.

#95 - 03/20/2024 06:47 AM - Constantin Asofiei

Alexandru, I agree, the change is in 7323b rev 15075.

#96 - 03/20/2024 07:36 AM - Greg Shah

- Status changed from Review to Internal Test

What testing is needed?

#97 - 03/20/2024 10:50 AM - Constantin Asofiei

Greg Shah wrote:

> What testing is needed?

I've tested with the #8363 app and there is no regression in performance terms. Functionally the change looks good.

#98 - 03/20/2024 11:05 AM - Greg Shah

- Status changed from Internal Test to Merge Pending

You can merge to trunk now.

#100 - 03/21/2024 08:41 AM - Alexandru Lungu

Constantin, unless you started the process, I will prepare for merge in ~10 mins.

#101 - 03/21/2024 08:43 AM - Constantin Asofiei

Branch 7323b was merged into trunk as rev. 15075 and archived.

#102 - 03/21/2024 08:43 AM - Alexandru Lungu

- Status changed from Merge Pending to Test

Got it!

Files

7323-JMX.patch	8.8 KB	07/04/2023	Radu Apetrii
7323-JMX.patch	8.95 KB	07/04/2023	Radu Apetrii
7323-JMX-before.patch	35.8 KB	07/20/2023	Radu Apetrii