Database - Feature #7382

Check performance of delete from vs drop table in H2

05/25/2023 08:05 AM - Alexandru Lungu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Alexandru Donica	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Database - Bug #7351: Reduce SQL query string size of an INSERT IN...	**Closed**

History

#1 - 05/25/2023 08:11 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Dănuț Filimon

This task targets a huge performance difference between DROP TABLE and DELETE FROM in H2 (#7351-6).
The point here is to clear some internal tables instead of dropping to maximize the use of psCache (not needing to recompile similar statements).
Please check #7351-5 for the use of DELETE instead of DROP.

I agree that a DROP simply unregisters a table from the table-map so the GC will do its job afterwards comparing to a DELETE FROM which needs to truncate indexes (eventually call triggers). However, I still expect a closer performance between these two, especially for tables with ~3 columns, no indexes and no triggers.

Please look into DELETE FROM implementation in H2 and try to short-circuit as much as possible.
Lastly, in FWD we usually use delete from tt where tt.multiplex = ?. Unless we plan to resolve this with direct-access, we should also focus optimizing such DELETE.

#2 - 05/29/2023 05:19 AM - Alexandru Lungu

- Related to Bug #7351: Reduce SQL query string size of an INSERT INTO SELECT FROM added

#3 - 06/27/2023 04:33 AM - Alexandru Lungu

This should be pending - #7459 may be a better way to fix this.

#4 - 07/26/2023 10:10 AM - Constantin Asofiei

There are cases where the temp-table is not really multiplexed across the FWD session - only one 'instance' of this temp-table exists. And a delete from tt1 where _multiplex = ? although targets the entire table, H2 will still remove the rows one-by-one - which adds the overhead of updating all the indexes when each row is removed.

I'm thinking maybe in H2's Delete.update we can short-circuit this code:

```
            int rowScanCount = 0;
            for (rows.reset(); rows.hasNext();) {
```

```
                if ((++rowScanCount & 127) == 0) {
                    checkCanceled();
                }
                Row row = rows.next();
                table.removeRow(session, row);
                session.log(table, UndoLogRecord.DELETE, row);
            }
```

and just empty all the indexes (including the ScanIndex) if the retrieved rows is the entire table (maybe add a PageStoreTable.clear() method).

#5 - 08/04/2023 06:51 AM - Alexandru Lungu

From Constantin's solution, I think we can just generate delete from tt1 when we know this as the last multiplex is use, right? Otherwise, FWD-H2 is not able to identify that the current delete will in fact delete all.
Danut, please address this.

#6 - 08/14/2023 02:40 AM - Constantin Asofiei

Alexandru, I didn't mean to change the DELETE SQL. What I meant in FWD-H2 is to check if the number of selected records which are to be deleted is the number of total records in the table. Can this be found easily?

#7 - 08/15/2023 03:45 AM - Dănuț Filimon

Constantin Asofiei wrote:

> What I meant in FWD-H2 is to check if the number of selected records which are to be deleted is the number of total records in the table. Can this be found easily?

In H2, we have access to the Table instance of the target when deleting records. From what I've looked into, it is possible to use Table.scanIndex to get the total number of rows of the target using ScanIndex.getRowCount(Session session). The problem is that we don't know the number of the records we want to delete. I've looked into the targetTableFilter for any useful information and I eventually compared the data between two delete statements (with and without conditions). The conclusion is that if the Delete.condition is null and there is no LIMIT expression being used, all the records should be deleted. In FWD, most delete statements use where multiplex = ? so the suggestion made by Alexandru in #7382-5 is a good starting point if we want to approach the change this way. I actually think that the delete statement should be shortened either way when it's the last multiplex in use.

Another and much simpler way would be to check if the numbers of rows is equal to the ScanIndex.rowCount right before iterating and removing the rows one by one. We can optimize this case specifically as you suggested in #7382-4, by emptying all the indexes.

#8 - 11/01/2023 08:53 AM - Alexandru Lungu

- Assignee changed from Dănuț Filimon to Radu Apetrii

With the new fwd-h2-1.33-trunk, FWD temp-tables have a special scan index that is able to distinguish between multiplex values (MultiplexedScanIndex). We can detect if only one multiplex exists using lastBatch. If the map has only one entry, it means that there is only one multiplex in scope and the table can be truncated instead of cleared one-by-one.

I will reassign this to you, Radu, as Danut is busy with way more other performance items right now.

#9 - 11/01/2023 10:31 AM - Alexandru Lungu

Just found out TRUNCATE TABLE tt in H2. I was suspecting that such command should have existed, but now it is confirmed :) My point is that we can use truncate instead of delete from tt where ... if we know there is one single multiplex used (based on inUseMPIDs). Constantin, I know that you mentioned that a solution on the FWD-H2 side is recommended, but as long as we have such information on FWD, can't we just do a conditional to check inUseMPIDs and eventually generate truncate. Is is cleaner than extending FWD-H2? I think it would be also faster, as we already know that there is one single multiplex in FWD. In FWD-H2, we need to infer that with some overhead.

#10 - 11/01/2023 10:40 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Just found out TRUNCATE TABLE tt in H2.

Isn't this the same as DELETE FROM tt without any WHERE clause? If so, we can just emit that if there is a single multiplex ID in use, **and also** we target that multiplex ID.

#11 - 11/01/2023 10:45 AM - Alexandru Lungu

Constantin Asofiei wrote:

> Alexandru Lungu wrote:
>
> > Just found out TRUNCATE TABLE tt in H2.
>
> Isn't this the same as DELETE FROM tt without any WHERE clause? If so, we can just emit that if there is a single multiplex ID in use, **and also** we target that multiplex ID.

Well, H2 documentation explicitly states that truncate is faster than DELETE without where clause :) It is in fact truncating the indexes in the implementation (data = null and that is all). The difference is that TRUNCATE is closing an open transaction. But this can be solved by adding a TRANSACTIONAL keyword, just like we did with create table, drop table, etc.

#12 - 11/06/2023 07:46 AM - Radu Apetrii

I put myself up to date with the discussion here and I created 7382a. I'll start working on this now.

#13 - 11/06/2023 07:49 AM - Constantin Asofiei

Alexandru Lungu wrote:

> The difference is that TRUNCATE is closing an open transaction. But this can be solved by adding a TRANSACTIONAL keyword, just like we did with create table, drop table, etc.

I assume that this will preserve tx rollback support from within H2?

#14 - 11/06/2023 08:15 AM - Alexandru Lungu

Constantin Asofiei wrote:

> Alexandru Lungu wrote:
>
> > The difference is that TRUNCATE is closing an open transaction. But this can be solved by adding a TRANSACTIONAL keyword, just like we did with create table, drop table, etc.
>
> I assume that this will preserve tx rollback support from within H2?

Good point. There is no undoLog operation for TRUNCATE. It can't eventually know what INSERT operations should be done to get back to the old state. This should be implemented. Basically, we need to iterate the scan key and add to undoLog all records. We can even short-circuit that for NO-UNDO tables.

As first step, we should get delete without multiplex working. After, we transform it to truncate after we implement it properly.

#15 - 11/17/2023 05:23 AM - Alexandru Lungu

- Assignee changed from Radu Apetrii to Eduard Soltan

Radu, please let me know if you managed to handle this to some extent. Otherwise, Eduard will continue working on this.

#16 - 11/17/2023 05:24 AM - Radu Apetrii

Alexandru Lungu wrote:

> Radu, please let me know if you managed to handle this to some extent. Otherwise, Eduard will continue working on this.

Unfortunately no. I was up to date with the discussion, but nothing more.

#17 - 01/05/2024 05:08 AM - Alexandru Lungu

- Assignee changed from Eduard Soltan to Alexandru Donica

Alexandru [ad], please provide a prototype for this task asap.

#18 - 01/09/2024 04:24 AM - Alexandru Donica

This is a patch of what I wrote so far. So from what I understand, instead of having a delete from tt where _multiplex = ?, we remove the where condition, if that is the only condition and if there is only 1 element in inUseMPIDs. I modified the method removeRecords() in TemporaryBuffer to not add the where condition if it is not necessary.
Do tell me if I missed/misunderstood anything.
The next step would be to put a truncate instead of delete? Or does this undoLog operation need to first be implemented for truncate? Or has it been implemented in the meantime/since?

```
=== modified file 'src/com/goldencode/p2j/persist/TemporaryBuffer.java'
--- old/src/com/goldencode/p2j/persist/TemporaryBuffer.java    2023-12-12 08:39:55 +0000
+++ new/src/com/goldencode/p2j/persist/TemporaryBuffer.java    2024-01-09 08:52:50 +0000
@@ -7969,25 +7969,35 @@
          query.append(" as ").append(dmoAlias);
       }

-      query.append(" where ");
+      boolean addWhere;
+      Map<Class<?>, Map<Integer, Integer>> map = local.inUseMPIDs;
+      Map<Integer, Integer> inUse = map.get(getDMOInterface());
+      addWhere = !forceDelete ||
+                 hasWhere ||
+                 !(isUse != null && inUse.size() == 1 && inUse.containsKey(multiplexID));

-      // apply multiplex filter and any additional conditions
-      if (hasWhere)
-      {
-         query.append("(");
-         if (hasOO) { list.append("("); }
-      }
-      query.append(mpidWhere);
-      additionalArgs.add(multiplexID);
-      if (hasOO)
-      {
-         list.append(dmoAlias).append(".").append(MULTIPLEX_FIELD_NAME).append("=?");
-         ooArgs.add(multiplexID);
-      }
-      if (hasWhere)
-      {
-         query.append(") and (");
-         if (hasOO) { list.append(") and ("); }
+      if (addWhere)
+      {
+         query.append(" where ");
+
+         // apply multiplex filter and any additional conditions
+         if (hasWhere)
+         {
+            query.append("(");
```

```
+            if (hasOO) { list.append("("); }
+         }
+         query.append(mpidWhere);
+         additionalArgs.add(multiplexID);
+         if (hasOO)
+         {
+            list.append(dmoAlias).append(".").append(MULTIPLEX_FIELD_NAME).append("=?");
+            ooArgs.add(multiplexID);
+         }
+         if (hasWhere)
+         {
+            query.append(") and (");
+            if (hasOO) { list.append(") and ("); }
+         }
       }
```

#19 - 01/09/2024 04:38 AM - Alexandru Lungu

> This is a patch of what I wrote so far. So from what I understand, instead of having a delete from tt where _multiplex = ?, we remove the where condition, if that is the only condition and if there is only 1 element in inUseMPIDs. I modified the method removeRecords() in TemporaryBuffer to not add the where condition if it is not necessary.

Please use 7382a. I will review the change there. Do extensive testing as this is a very invasive change.

> The next step would be to put a truncate instead of delete? Or does this undoLog operation need to first be implemented for truncate? Or has it been implemented in the meantime/since?

At a second thought, maybe we should simply optimize delete from tt construct to behave like truncate. Think like a short-circuit. I think the work on adding UNDO support for TRUNCATE and making it transactional way more complex than simply optimizing DELETE. You need to check that there is no WHERE, the table is NO-UNDO, no database triggers are defined, etc. In that case, use the truncate routine (truncate all indexes). 7382a_h2 already exists (just rebased it now to latest FWD-H2 trunk).

The main focus here is performance. Please set-up the large performance POC tests we have and see if how there 2-part changes are behaving. Also mind tracking how many times your optimization is hit (as a ratio: number of deletes wihtout multiplex / total number of deletes).

PS: also set-up the large regression tests application to ensure there are no obvious regressions at first (with the patch you suggest).

#20 - 01/09/2024 05:06 AM - Alexandru Lungu

Alexandru [ad], please refer to H2_Database_Fork for further insights.

#21 - 01/15/2024 10:17 AM - Alexandru Donica

There is a bug/regression somewhere regarding the patch I posted above. When I try to smoke test on another large app to, an error quickly appears stating "Cannot use a closed database connection", which appears at a point where the where condition in the query is removed, to optimize. But it seems it breaks something else, I have yet to find out where.
Maybe the condition to remove the where clause should be tighter.

#22 - 01/16/2024 09:43 AM - Alexandru Donica

Scratch that, there was a little bug, but one that would show a weird error related to db because it was happening in a class in h2 i modified.
So I started to run the POC tests to first see how often the 'fast delete' would be used compared to the normal delete, then I wanted to compare performance. But I noticed that the 'fast delete' would never get called during the POC tests, and AFAIK, if 'fast delete' would not be called in 1 iteration, it would not be called in any of the 100 iterations set to run.
However, in another large app, I did see the 'fast delete' being used, far less than the normal delete (like 4% of deletes were 'fast deletes' from the moment the server started, and while I opened some menus and roamed through the large app).
Is there a different way I should test this change?

#23 - 01/16/2024 10:13 AM - Alexandru Lungu

This is quite strange - there are many "one time" temp-tables used in the large POC (or this is what I recall). Please debug some of the POC code.

#24 - 01/17/2024 09:30 AM - Alexandru Donica

After some debugging, I found an error that I didn't know was thrown, which was happening because, when the classic delete, row by row, happens, the constraints are respected (probably constraints like cascade deletes), but when truncating, it does not automatically delete every other record from other tables that should be deleted in cascade. I believe this because the exception I was getting stated something like 'duplicates not allowed'.
And I added the condition, to 'fast delete' if there are no constraints (which would mean tables with extent fields would can't be 'fast deleted'). But with this new condition added, there are no 'fast deletes' happening in the POC tests. One of the conditions would always fail, either the table is undo or it has constraints, or something else.

#25 - 01/17/2024 09:34 AM - Greg Shah

As far as I know, the only cascading deletes that we would encounter would be for the non-expanded extents (which have related secondary tables). After we move to always expanding extents, this constraint would be gone.

Do other constraints exist? If not, then this again suggests high priority for eliminating non-expanded extents.

#26 - 01/17/2024 09:59 AM - Alexandru Donica

I modified the code, now I only check for referential constraints, as they are the ones causing problems (Some tables had other types of constraints, that should not cause issues). Now I noticed that ~1 in 7 Delete.update() calls is a 'fast delete'. I will update when I run the 100 tests again to see the average time.

#27 - 01/17/2024 11:43 AM - Alexandru Lungu

I really think this ratio will go up once we use expanded extents. There are many temp tables with extent fields. As they are non-expanded, it simply rules out your optimization.

Alexandru [ad], please focus on the time difference. If the preconditions of a "fast delete" are met, how much time it takes to do the classic delete (row

by row) vs truncate - all inside H2. Compute:

- Number of total deletes
- Number of times the preconditions of a "fast delete" are met
- Number of time the referential constraints rule out the "fast delete"
- Total and Avg. time for classic deletes
- Total and Avg. time for fast deletes
- Total and Avg. time of classic deletes that were optimized into fast deletes

#28 - 01/18/2024 03:25 AM - Alexandru Donica

Alexandru Lungu wrote:

- Total and Avg. time for classic deletes
- Total and Avg. time for fast deletes
- Total and Avg. time of classic deletes that were optimized into fast deletes

So, firstly measure time for the classic deletes, meaning without the optimized code? Or with the optimized code, but not counting in the fast deletes (which would not count ALL the deletes, but all the classic ones)?
Secondly, same question as above but for fast deletes, measure ONLY the fast deletes, or classic + fast?
And, deletes that were optimized into fast deletes, depending on the answer to the "Secondly," question, this question may just be a repeat. Or did you mean that I measure ONLY fast deletes here, and classic+fast for question above?

#29 - 01/18/2024 03:40 AM - Alexandru Lungu

Lets presume that A is the set of **all** deletes and C is the set of "candidate" deletes (the ones that **can** be optimized).
By classic I mean without your changes. By fast, I mean with your changes.

I rephrase:

1. (Old version) All deletes without any of your changes. (A in classic execution)
2. Only the deletes that are candidate for optimization, but were not optimized. (C in classic execution)
3. Only classic deletes that are not candidate for optimization (A \ C in classic execution)
4. Only the fast deletes that were candidate for optimization and were optimized. (C in fast execution)
5. (New version) Deletes that are not candidate are done classically, while candidates are optimized. (A \ C in classic execution + C in fast execution)

My goal:

- Compare 1. and 5. for overall improvement
- Compare 2. and 4. for net improvement of deletes we targeted
- Consider 3. to understand how much time is still spend on un-optimized deletes.

#30 - 01/18/2024 04:05 AM - Alexandru Donica

well 5 = 4 + 3. and 1 = 2 + 3.
(don't read this out of context)

#31 - 01/18/2024 09:32 AM - Alexandru Donica

1. 8591 ms for 939_203 deletes (classic execution, all deletes)
5. 7672 ms for 935_965 deletes (optimized execution, all deletes)

2. ~896 ms for 117_794 deletes (10% of total time for 12% of total deletes) (classic execution, only candidate deletes)
4. 162 ms for 117_282 deletes (2% of total time for 12% of total deletes) (optimized execution, only candidate deletes)

3. 7695 ms for 821_408 deletes (89% of total time for 88% of total deletes) (classic execution, only non-candidate deletes) (in the run with the optimized code, this took 7510 ms (97% of total time) for 818_683 deletes)

The number of deletes that would have been optimized if and only if there were no referential constraints is ~18_580, ~2% of total deletes.

Average time of a delete A \ C = 0.0093 ms per delete.
Average time of a delete C optimized = 0.0014 ms per delete.
Average time of a delete C non-optimized = 0.0076 ms per delete.
Average time of deletes A \ C + C-optimized = 0.0082 ms per delete.
Average time of deletes A \ C + C-non-optimized = 0.0091 ms per delete.

For 1 optimized run, average total time of 100 tests was 14554 ms, so a total of 1_455_400 ms.
For 1 non-optimized run, average total time of 100 tests was 14528 ms, so a total of 1_452_800 ms.
All the deletes for an optimized run counted as 0.52% of all execution time.
All the deletes for a non-optimized run counted as 0.59% of all execution time.

In conclusion, in theory it's a small improvement, but it's not visible in the total time of the 100 runs as I only ran 1 time for each case.

#32 - 01/23/2024 03:58 AM - Alexandru Lungu

- % Done changed from 0 to 50

I think we can go ahead with this improvement. Even if it is small, it looks like it is quite risk free and easy to implement. Please commit the changes in 7382a and 7382a_h2.

#33 - 01/23/2024 06:29 AM - Alexandru Lungu

Alexandru [ad], I also notices a slight performance improvement possibility in Update. There is a setClauseCols boolean array. true means that the columns should be updated, false means that it should not be updated. There are a large number of update statements that update a single columns. The bad part is that Update iterated all columns of the table. This is particularly bad for tables with >100 columns. This may not be much, but please attempt to replace setClauseCols with BitSet. Set the corresponding bit in setAssignment. Iterate only st bits using nextBitSet.

This is not related to #7382 per-se, but may be integrated at once. Please profile this separately and compute some performance results.

#34 - 01/23/2024 08:36 AM - Alexandru Donica

Very well I will try to improve Update, but before that, should the branch 7382a be a copy of the trunk, or of branch 7156b (with which I tested)? Now, it is a copy of trunk.

#35 - 01/23/2024 09:22 AM - Alexandru Lungu

7382a is branched for trunk, ready to be merged. If you want to test, you need to manually port the changes to 7156b. At some point, 7156b will be rebased with trunk.

#36 - 01/25/2024 05:54 AM - Alexandru Donica

- Status changed from WIP to Review

With the changes you mentioned, the amount of time spent iterating in 100 runs of the performance tests went down from 2500 ms to 2000 ms. Not much of an improvement overall, but a 20% improvement in that area of code.

I have commited all the changes, for Update and Delete, in branch 7382a_h2 rev no. 39, and branch 7382a rev no. 14916.

#37 - 01/31/2024 04:37 AM - Alexandru Lungu

- Status changed from Review to Internal Test

I am OK with the changes; merged 7382a_h2 to FWD-H2 trunk as rev. 40. Archived 7382a_h2.

Alexandru [ad], please pick up the latest trunk (rev. 40) and run the full regression tests of the large customer application. Just check-out, build and copy the archive in deploy/lib and remove the old FWD-H2 artifact from there.

If they pass, we will upload fwd-h2-1.40-trunk.jar to the public repository.

#38 - 02/01/2024 08:47 AM - Alexandru Donica

- % Done changed from 50 to 100

I ran the tests, and there were no more tests that failed compared to the baseline.

#39 - 02/01/2024 05:04 PM - Alexandru Lungu

- Status changed from Internal Test to Test

That is great. These changes are already in the H2 trunk - I will move this into Test.

#40 - 02/02/2024 09:29 AM - Greg Shah

Don't we need to get 7382a into trunk to take advantage of the improvements?

#41 - 02/02/2024 11:52 AM - Alexandru Lungu

Branch 7382a was merged into trunk as rev 14966 and archived.

#42 - 02/02/2024 11:55 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Branch 7382a was merged into trunk as rev 14966 and archived.

Doesn't this require update to fwd-h2 rev 40?

#43 - 02/02/2024 11:59 AM - Alexandru Lungu

Right, it does! I uploaded devsrv01:/tmp/fwd-h2-1.40-trunk.jar. Please upload to public repository.
We can then upgrade H2 directly in trunk.

#44 - 02/02/2024 12:02 PM - Constantin Asofiei

Alexandru Lungu wrote:

> Right, it does! I uploaded devsrv01:/tmp/fwd-h2-1.40-trunk.jar. Please upload to public repository.

Done.

> We can then upgrade H2 directly in trunk.

Please go ahead. Don't forget the Refs #7382 in the commit message.

#45 - 02/02/2024 12:04 PM - Alexandru Lungu

Done