Database - Support #4702

write temp-table performance test cases

06/29/2020 05:11 PM - Eric Faulhaber

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

Description

Related issues:

Related to Database - Bug #4057: h2 performance degradation	**New**
Related to Database - Support #4701: try to improve H2 transaction commit per...	**Closed**
Related to Database - Bug #4703: investigate whether performance of TempTable...	**Hold**
Related to Database - Support #6679: H2 general performance tuning	**Internal Test**

History

#1 - 06/29/2020 05:12 PM - Eric Faulhaber

- Related to Bug #4057: h2 performance degradation added

#2 - 06/29/2020 05:12 PM - Eric Faulhaber

- Related to Support #4701: try to improve H2 transaction commit performance added

#3 - 06/29/2020 07:43 PM - Eric Faulhaber

We need a set of temp-table related test cases in support of issues #4057 and #4071. The tests need to stress H2 as the backing database of our temp-table and dirty share implementations in FWD.

The tests should explore the major facets of CRUD operations, preferably in separate tests or test parts which can be independently measured/profiled. They should be long-running enough to allow the user to capture adequate profiling data for analysis, giving the JVM a chance to "warm up". We do not need a harness to run all of them at this point. That may come later, but for now, we will want to run them independently to best know what type of operations we are stressing.

The set should include heavy, sustained temp-table use of:

- reads;
- writes;
- updates;
- deletes;
- many small transactions;
- one large transaction;
- no transaction (such that records must be auto-committed).

I would like at least one test which "thrashes" JDBC connections. That is, the test should allow the temp-table ORM session to close between iterations of its CRUD operation(s), so that a new connection needs to be made on each iteration. I believe this will be achieved by putting the temp-table use in one external procedure which is called many times from another.

The temp-tables should use a variety of data types, but they don't need to be anything exotic. 3-5 fields should be adequate. It is important that the tables tested have a variety of indexes: multiple unique, multiple non-unique, a combination, or none at all. Perhaps this is best achieved by using include files to define the same tables with different indices. The index composition will change how we hit H2 from FWD.

The temp-tables should be loaded with a reasonable number of rows (say, 1000).

The goal of these tests is not to find functional problems. They should be straightforward in what they do and focus on stressing the performance of "vanilla" temp-table operations.

Given these requirements, try to keep the set as simple as possible to set up, convert, and run. There should be no requirement for a persistent schema; all temp-tables should be defined in the 4GL code.

#4 - 06/30/2020 05:51 AM - Eric Faulhaber

- Related to Bug #4703: investigate whether performance of TempTableDataSourceProvider can be improved added

#5 - 06/30/2020 08:17 AM - Greg Shah

Perhaps it makes sense to have a test that has some temp-tables with large numbers of columns and uses buffer-copy to copy many records between the tables?

#6 - 06/30/2020 08:58 AM - Stanislav Lomany

I'm think there will be tables which will have the same set of fields:

1. integer
2. char
3. double
4. date
5. logical

There will be four tables with corresponding indexes:

1. one unique index
2. one non-unique index
3. one unique index and one non-unique index
4. no indexes

There will be queries responsible for record iteration. Iteration order will be:

1. matching a table index
2. unmatching table indexes

Actions will be:

1. create records (record order relatively to the index will be deterministic, but not sequential).
   1. autocommit
   2. many single-record transactions
   3. one single transaction
2. read records (load field values into variables)
3. update records
   1. autocommit
   2. many single-record transactions
   3. one single transaction
4. delete records (bulk)
5. delete all records (individually)
   1. autocommit
   2. many single-record transactions
   3. one single transaction

Is that good for a start?

#7 - 06/30/2020 01:06 PM - Eric Faulhaber

Stanislav Lomany wrote:

> [...]
> Is that good for a start?

Yes.

#8 - 06/30/2020 01:09 PM - Eric Faulhaber

Greg Shah wrote:

> Perhaps it makes sense to have a test that has some temp-tables with large numbers of columns and uses buffer-copy to copy many records between the tables?

I want to be careful we don't creep the scope of this set of tasks. I do think that makes sense for #3958. But what's the rationale, in terms of stressing H2?

#9 - 06/30/2020 02:14 PM - Greg Shah

It is a common case in which a lot of time is likely to be spent in real applications. Any server-side stuff is going to be heavy on this and even many GUI client applications will be written to call into appserver for all their data needs.

On top of that, "wider" tables (with lots of columns) may have different performance characteristics than narrow tables. The combination of both of these seems quite useful to ensure we don't miss something obvious.

#10 - 07/01/2020 08:22 AM - Stanislav Lomany

I've committed read and update testcases to testcases/uast/h2_performance. The remaining testcases will follow.

#11 - 07/01/2020 01:47 PM - Eric Faulhaber

Looks good so far.

In addition to the OPEN QUERY use you have in the read tests, please add some stressful use of FOR EACH and FIND, to make the tests even more representative of much of the old 4GL code we find "in the wild". Thanks.

#12 - 07/02/2020 04:21 AM - Igor Skornyakov

Stanislav.
Which of your tests is supposed to open/close multiple JDBC Connections?
Thank you.

#13 - 07/02/2020 05:53 AM - Stanislav Lomany

> Which of your tests is supposed to open/close multiple JDBC Connections?

Igor, I'm not managing connection. I suppose that is a part of your task.
BTW, I just committed more testcases for record creation and deletion.

#14 - 07/02/2020 06:12 AM - Igor Skornyakov

Stanislav Lomany wrote:

> Igor, I'm not managing connection. I suppose that is a part of your task.
> BTW, I just committed more testcases for record creation and deletion.

I see. Thank you, Stanislav.

#15 - 07/02/2020 06:28 AM - Eric Faulhaber

Eric Faulhaber wrote:

> I would like at least one test which "thrashes" JDBC connections. That is, the test should allow the temp-table ORM session to close between iterations of its CRUD operation(s), so that a new connection needs to be made on each iteration. I believe this will be achieved by putting the temp-table use in one external procedure which is called many times from another.

Whoever writes the connection-thrashing test, I think the above technique is the way to achieve it.

#16 - 07/02/2020 05:59 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Eric Faulhaber wrote:
>
> > I would like at least one test which "thrashes" JDBC connections. That is, the test should allow the temp-table ORM session to close between iterations of its CRUD operation(s), so that a new connection needs to be made on each iteration. I believe this will be achieved by putting the temp-table use in one external procedure which is called many times from another.

Whoever writes the connection-thrashing test, I think the above technique is the way to achieve it.

I've tested calling one of Stanislav's tests (with minor changes to suppress output) as an external program:

```
DO i = 1 TO 1000000:
    RUN h2_performance/perf-create1.p.
END.
```

With FWD it runs several orders of magnitude faster than with 4GL but the connections are created only at the server startup. Which modifications are required to test "connection trashing"?
Thank you.

#17 - 07/02/2020 07:36 PM - Stanislav Lomany

- Status changed from New to Review

I've committed all of the tests, including FOR EACH, FIND and BUFFER-COPY tests.

#18 - 07/03/2020 04:16 PM - Eric Faulhaber

FYI, I added some missing EOL markers to perf-run-all-silent.p and committed the fix.

#19 - 07/04/2020 05:59 PM - Eric Faulhaber

There is an issue running the update tests with 4011b/11544. Looks like we are getting into an infinite loop based on the dynamic updates within the FOR loops. I'll look at this.

Stas, in addition to use with #4057 and #4701, these are good, general purpose benchmark tests for temp-table performance. Could you please add some additional text to the output, so it is clear what each row being displayed represents? Also, the qnum and tres headings are a bit cryptic. Could you make these labels a bit more verbose/meaningful, please? Thanks.

#20 - 07/09/2020 01:08 PM - Stanislav Lomany

Results so far:

1. Looks like "UPDATE using a query" test goes to infinite cycle in 4011b (H2 197) if the number of records exceeds some number (10 records works fine, 100 doesn't work).
2. Although 4011b (H2 197) is ~50% faster than trunk in these tests, "UPDATE using FOREACH" test is ~250% *slower*.
3. To run H2 200 the following code was commented out in DatabaseManager, because there is no org.h2.engine.Constants.getFullVersion() function:

   ```
   log.log(Level.INFO, "Using H2 database version " + H2Helper.getH2Version());
   ```

4. Right now I'm blocked with the following exception on server startup (fwd-h2-1.4.200-20200709.jar):

```
com.goldencode.p2j.cfg.ConfigurationException:  Initialization failure
    at com.goldencode.p2j.main.StandardServer.hookInitialize(StandardServer.java:2083)
    at com.goldencode.p2j.main.StandardServer.bootstrap(StandardServer.java:999)
    at com.goldencode.p2j.main.ServerDriver.start(ServerDriver.java:483)
    at com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:444)
    at com.goldencode.p2j.main.ServerDriver.process(ServerDriver.java:207)
    at com.goldencode.p2j.main.ServerDriver.main(ServerDriver.java:860)
Caused by: java.lang.RuntimeException: Error initializing persistence services
    at com.goldencode.p2j.persist.Persistence.initializeInstance(Persistence.java:3968)
    at com.goldencode.p2j.persist.PersistenceFactory.getInstance(PersistenceFactory.java:157)
    at com.goldencode.p2j.persist.DatabaseManager.initialize(DatabaseManager.java:1607)
    at com.goldencode.p2j.persist.Persistence.initialize(Persistence.java:881)
    at com.goldencode.p2j.main.StandardServer$11.initialize(StandardServer.java:1244)
    at com.goldencode.p2j.main.StandardServer.hookInitialize(StandardServer.java:2079)
    ... 5 more
Caused by: java.util.ConcurrentModificationException
    at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)
    at java.util.ArrayList$Itr.next(ArrayList.java:851)
    at org.hibernate.engine.jdbc.internal.LogicalConnectionImpl.close(LogicalConnectionImpl.java:208)
    at org.hibernate.engine.jdbc.internal.JdbcCoordinatorImpl.close(JdbcCoordinatorImpl.java:141)
    at org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl.close(TransactionCoordinatorIm
pl.java:276)
    at org.hibernate.internal.SessionImpl.close(SessionImpl.java:352)
    at com.goldencode.p2j.persist.dialect.H2Helper.executeDDL(H2Helper.java:531)
    at com.goldencode.p2j.persist.dialect.H2Helper.createFunctionAliases(H2Helper.java:427)
    at com.goldencode.p2j.persist.dialect.H2Helper.prepareDatabase(H2Helper.java:201)
    at com.goldencode.p2j.persist.Persistence.initializeInstance(Persistence.java:3948)
    ... 10 more
```

#21 - 07/09/2020 02:03 PM - Adrian Lungu

Stanislav, I will try to reproduce the exception and see if it related to fwd-h2-1.4.200-20200709.jar.
I remember some changes which should be done to work with 1.4.200, beside #4702-20 point 3. Check #4701-16 point 2. I don't think that the mvcc and multi_threaded issue is related to this case, but should be taken in consideration anyways.

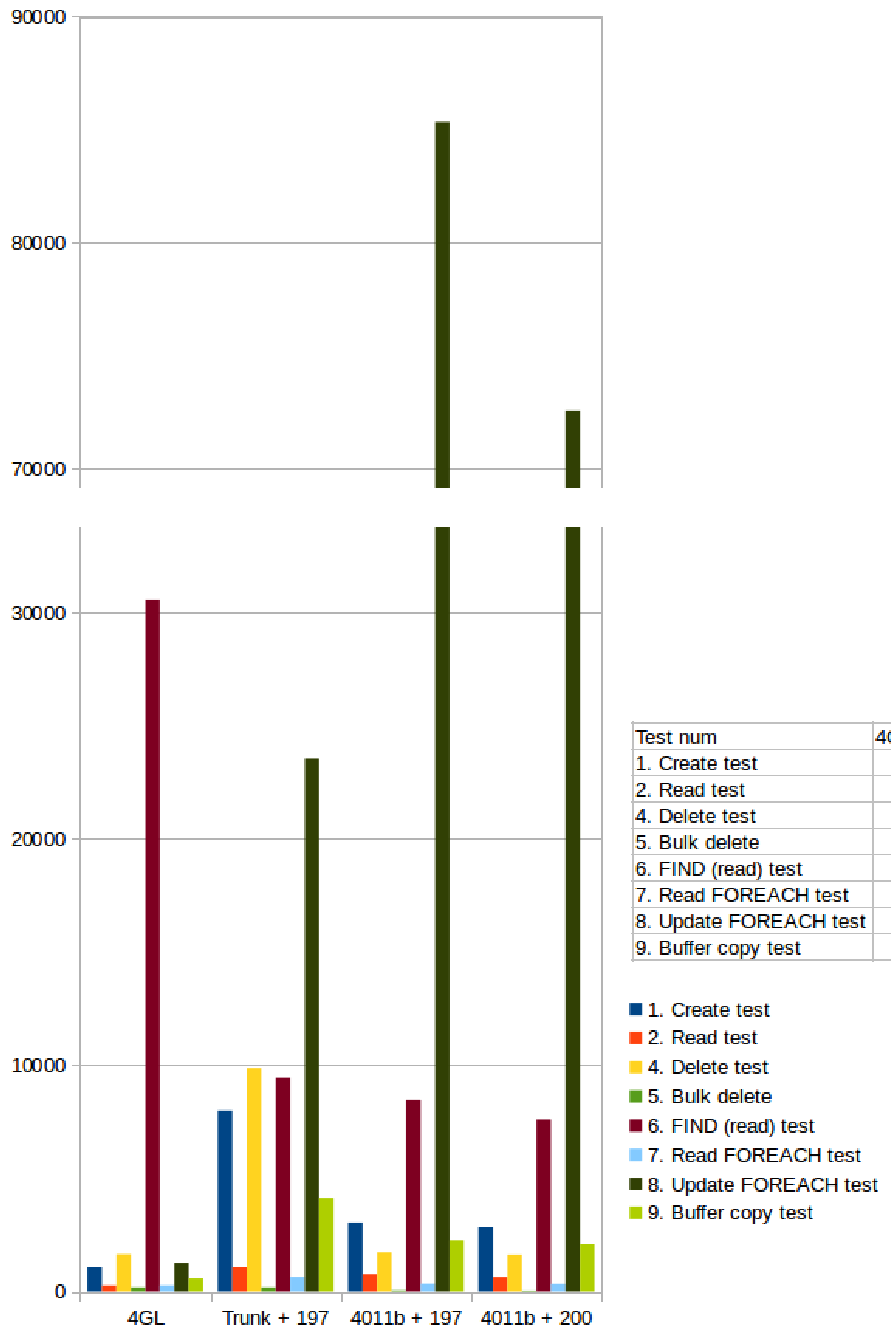
#22 - 07/09/2020 04:44 PM - Stanislav Lomany

I made some changes to FWD to make H2 200 run. Starting testing.

#23 - 07/09/2020 06:45 PM - Stanislav Lomany

- File h2-200.png added

Testing results: each test is faster on H2 200 (fwd-h2-1.4.200-20200709) than on H2 197, in average 13% faster.
Test number 3 is missing because it doesn't work properly with 4011b.



Test num	4GL	Trunk + 197	4011b + 197	4011b + 200
1. Create test	1062	7991	3052	2843
2. Read test	259	1064	770	640
4. Delete test	1638	9882	1728	1619
5. Bulk delete	184	185	50	38
6. FIND (read) test	30582	9443	8466	7608
7. Read FOREACH test	260	645	336	326
8. Update FOREACH test	1261	23562	85348	72577
9. Buffer copy test	576	4136	2272	2091

#24 - 07/10/2020 05:18 PM - Stanislav Lomany

- File h2-200.diff added

For the case I'm attaching changes I made to 4011b in order to run H2 200 which are:

1. Constants.getFullVersion() replaced with Constants.FULL_VERSION.
2. Removed references to MULTI_THREADED and MVCC parameters.

#25 - 07/10/2020 06:15 PM - Eric Faulhaber

Stanislav Lomany wrote:

> For the case I'm attaching changes I made to 4011b in order to run H2 200 which are:
>
> 1. Constants.getFullVersion() replaced with Constants.FULL_VERSION.
> 2. Removed references to MULTI_THREADED and MVCC parameters.

Please go ahead and commit these changes to 4011b, since we are moving to v1.4.200 now for this branch.

The removal of MULTI_THREADED (from H2, not 4011b) is a little concerning. Do you have any more information on what this actually means? Is it the default now, or did they remove the support? Or something else...?

#26 - 07/13/2020 02:24 AM - Eric Faulhaber

Eric Faulhaber wrote:

> Please go ahead and commit these changes to 4011b, since we are moving to v1.4.200 now for this branch.

I committed these changes in 4011b/11555.

#27 - 07/13/2020 03:38 AM - Stanislav Lomany

> The removal of MULTI_THREADED (from H2, not 4011b) is a little concerning. Do you have any more information on what this actually means? Is it the default now, or did they remove the support? Or something else...?

```
MULTI_THREADED setting is removed, MVStore engine is always multi-threaded, PageStore engine is always single-threaded
```

#28 - 07/13/2020 03:44 AM - Eric Faulhaber

Stanislav Lomany wrote:

> PageStore engine is always single-threaded

Do they claim this? That doesn't seem right. I've made edits to that code and there is an awful lot of synchronization code in that engine.

BTW, did you do any testing with the MVStore engine? In my previous experience, FWD was slower with MVStore, but I had not tested it with v200 before.

#29 - 07/13/2020 03:49 AM - Stanislav Lomany

> Do they claim this?

They do:
https://webcache.googleusercontent.com/search?q=cache:OZQ8T12NI00J:https://github.com/h2database/h2database/releases+&cd=1&hl=ru&ct=clnk&gl=ru

http://www.h2database.com/html/changelog.html (issue 2150)

#30 - 07/13/2020 03:58 AM - Stanislav Lomany

> BTW, did you do any testing with the MVStore engine? In my previous experience, FWD was slower with MVStore, but I had not tested it with v200 before.

No. But it may make sense to test both engines. Here's a comment about performance decrease in PageStore after MVCC was removed and MVStore performace increase: http://h2-database.66688.n3.nabble.com/Performance-issues-with-PageStore-databases-td4035568.html

#31 - 07/13/2020 04:08 AM - Eric Faulhaber

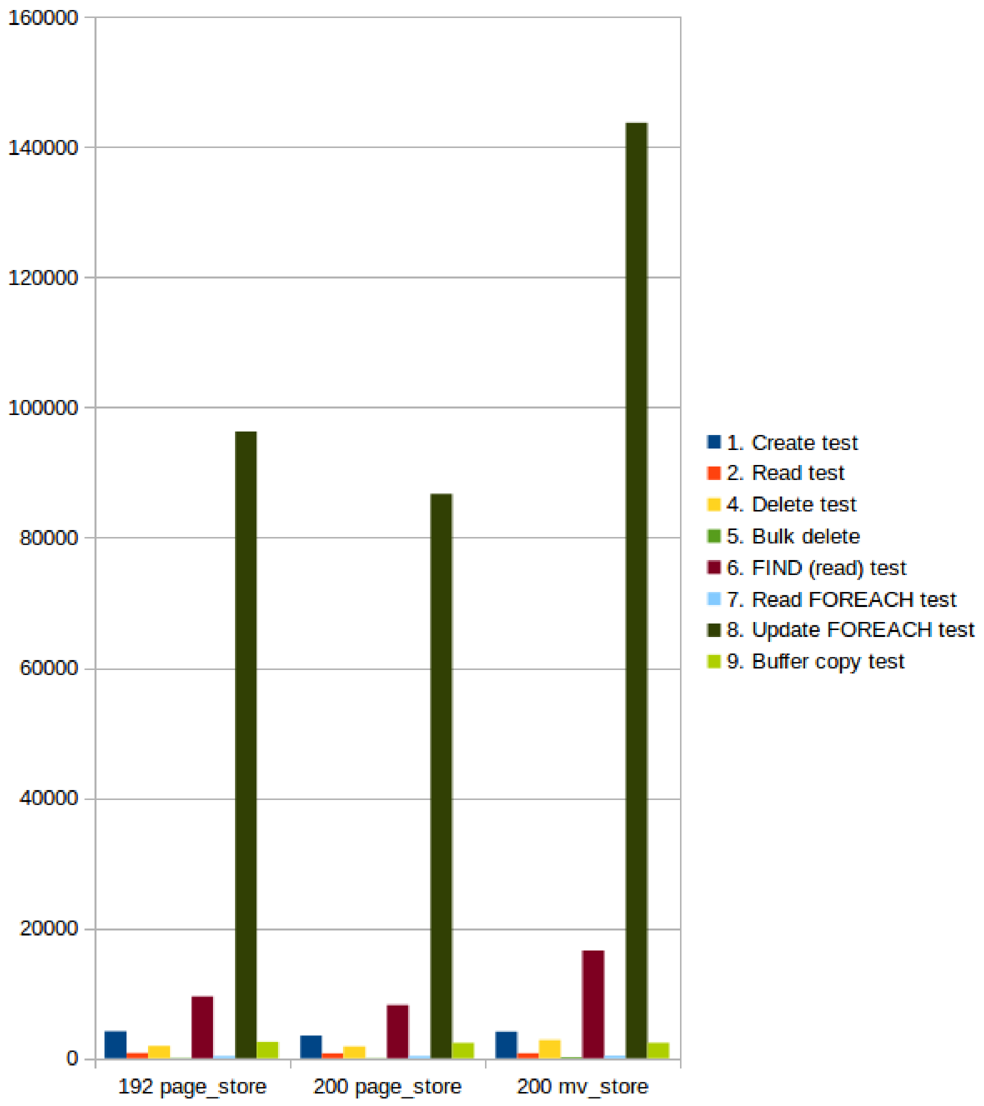
Yes, I'd like to see a comparison of your tests with:

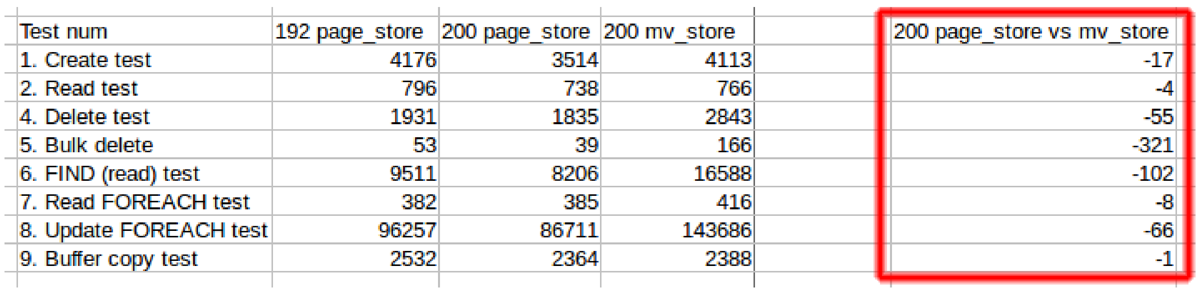
- 1.4.197 (patched) PageStore
- 1.4.200 (patched) PageStore
- 1.4.200 (patched) MVStore

#32 - 07/13/2020 02:00 PM - Stanislav Lomany

- File mv_store.png added

According to test results, MVStore (mv_store=true) is significantly slower than PageStore (mv_store=false). The red square contains time differences for specific tests in percents.





Test num	192 page_store	200 page_store	200 mv_store		200 page_store vs mv_store
1. Create test	4176	3514	4113		-17
2. Read test	796	738	766		-4
4. Delete test	1931	1835	2843		-55
5. Bulk delete	53	39	166		-321
6. FIND (read) test	9511	8206	16588		-102
7. Read FOREACH test	382	385	416		-8
8. Update FOREACH test	96257	86711	143686		-66
9. Buffer copy test	2532	2364	2388		-1

#33 - 07/13/2020 02:02 PM - Stanislav Lomany

In the chart it should be "197", not "192".

#34 - 07/17/2020 10:28 AM - Stanislav Lomany

The H2 performance testcases have been committed into the new testcases project as rev 668. They have been removed from the old testcases project.

#35 - 07/17/2020 06:37 PM - Stanislav Lomany

I've added section how to run performance testcases from the new testcases project:
https://proj.goldencode.com/projects/p2j/wiki/Performance_Testing_the_H2_Database#section-5

#36 - 08/17/2022 10:57 AM - Alexandru Lungu

- Related to Support #6679: H2 general performance tuning added

#37 - 01/23/2024 04:33 PM - Eric Faulhaber

- Status changed from Review to Closed

- % Done changed from 0 to 100

I think this task has served its purpose in guiding our performance work with H2.

Files

h2-200.png	48.5 KB	07/09/2020	Stanislav Lomany
h2-200.diff	2.83 KB	07/10/2020	Stanislav Lomany
mv_store.png	46.7 KB	07/13/2020	Stanislav Lomany