Database - Feature #7062

Direct access to H2 internals to solve basic queries

01/23/2023 05:38 AM - Alexandru Lungu

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

Description

History

#1 - 01/23/2023 06:23 AM - Alexandru Lungu

I've done slim experiments with the H2 internals to understand how easy/fast is to interface directly with H2. I have the following comparative avg. results for the fetching of all records by recid from a table with 1,000,000 records:

- 2,800 ms for using a static SQL for each record: these are used in FWD if the FQL is statically converted.
- 800 ms for using a single prepared statement: this is used in FWD if there are query parameters and the prepared statement is cached
- 300 ms for direct use of the primary key (tree) index: this is not in FWD, but can be used if FQLPreprocessor detects that this is a find by rowid type of query.
- 30 ms for direct use of scan index key look-up: this is not in FWD and requires a bit of work. This kind of interaction requires the record key. This key is basically the index of the record inside the scan index array. Maybe we can keep a server-side mapping between multiplex/recid and H2 key for faster look-up?

All in all, the direct access, as expected, is faster as it doesn't need to interface with JDBC through SQL that needs to be generated, parsed, cached, planned, executed, etc. I don't promote this kind of interaction on a general scale. However, there are many basic queries which are trashed between different layers to accomplish very simple tasks (e.g. retrieving a record by its recid).

For this matter, I suggest using some sort of EmbeddedFWDDriver inside the H2 engine as an interface for very simple tasks, which can be accomplished directly on the database store where there is no need of locking, transactions, complex planning, etc.
Below is a snippet I used for testing, although it may need some improvements:

```
      JdbcConnection conn = (JdbcConnection) DriverManager.getConnection("jdbc:h2:mem:;mv_store=false;");
      Session session = (Session) conn.getSession();

      // direct access
      Table t = session.findLocalTempTable("TT");
      Index idx = t.getUniqueIndex();
      Cursor cursor = idx.find(session, new SimpleRow(new Value[] { ValueInt.get(1) }), null);
      if (cursor.next())
      {
         Row r = cursor.get();
         int v = r.getValue(0).getInt();
      }
```

Once we can get a better grip on this kind of interaction, we can extend it to aid higher level of representations from FWD:

- FIND tt WHERE recid(tt) = 2.
- FIND FIRST/LAST tt. is basically the left-most/right-most node in the used index.
- FOR/PRESELECT tt WHERE recid(tt) = 2.
- find with unique indexed field

From the implementation point of view, this can go as a special kind of Results implementation in FWD, one that simply uses EmbeddedFWDDriver.

#2 - 01/23/2023 02:52 PM - Greg Shah

This is really interesting. For temp-tables (and using H2 for a permanent database), it is conceptually close to the way the 4GL accesses records in shared memory mode.

Would this be used when we have a projection query that returns the primary key for a record (or list of primary keys for a list of records)?

#3 - 01/24/2023 05:16 AM - Alexandru Lungu

Greg Shah wrote:

> This is really interesting. For temp-tables (and using H2 for a permanent database), it is conceptually close to the way the 4GL accesses records in shared memory mode.

Exactly. However, I didn't thought of H2 permanent databases in the first place. I had the private temp-databases in my head as it was easier to think of an environment without locking and transactions. When we consider shared temp-database and H2 permanent database, we need to do some extra effort in ensuring that the direct access honors the locks.

> Would this be used when we have a projection query that returns the primary key for a record (or list of primary keys for a list of records)?

AFAIK, projection queries are only for the permanent database (or not?). So this use-case falls into the scenario in which we implement direct access for H2 permanent database, which I didn't had in mind at first.
Now that you put H2 direct access and projection queries in the same sentence, I am thinking even of PreselectQuery or AdaptiveQuery doing projections on the **temp-database** and let EmbeddedFWDDriver do the hydration internally when needed. This may overlap with #7061. However, this case is really new to me now and I don't have a strong understanding of it yet.

Just to make this clear, my initial idea was to resolve faster the temp-table record look-up by recid. If we encounter something like FIND tt WHERE recid(tt) = 2, we avoid building an SQL, but directly retrieve the record from the primary key index.

#4 - 01/27/2023 07:58 AM - Alexandru Lungu

- % Done changed from 0 to 10

- Status changed from New to WIP

- Assignee set to Alexandru Lungu

I've done some consistent implementation for the FIND tt WHERE recid(tt) = 2 use-case. However, I am facing some limitations I am trying to overcome:

- (*HYDARTION*) We still need the hydration step as we convert the H2 record representation into our ORM representation. There is a lot of code

for this in FWD, but it is highly coupled with ResultSet. Basically, our DataHandler interface forces the extraction of data through a ResultSet, so using the JDBC interface. At this point, I tried to implement a more general mean of hydrating:
  - SQLHydrator used for hydrating records through a ResultSet. This is what we do now.
  - DirectAccesHydrator used for hydrating records through a H2 internal representation (org.h2.value.Value[] or org.h2.result.Row). This is new to us, but is very similar to a ResultSet as we can retrieve certain kind of data through an index.
  - Because of the high complexity of refactoring all DataHandler in FWD, I am postponing DirectAccesHydrator. For now, I am wrapping the org.h2.result.Row into a dummy ResultSet to benefit from the current hydration implementation in FWD.
- (*SELECT*) There are H2 tables which have computed columns (for datetime fields, extents, indexed character fields, etc.).
  - The best is to skip selection of fields if we have no computed columns (much like we do with SQL wildcard in #7060)
  - For the other cases, we either avoid direct access or we send the fields to EmbeddedFWDDriver. From there, we can do the field selection at the H2 level. However, this is the same thing an SQL does, so I won't expect much benefit from direct access only for some fields.
  - At best, we can implement the #7060 suggestion of adding a HIDDEN keyword for H2 temp-tables fields which shouldn't be accessed with wildcard or direct access.
- (*WHERE*) The index look-up we do inside H2 should rely on the fact that a (multiplex, recid) index exists (eventually as primary key).
  - For this matter, **I have a question**: is (multiplex, recid) always a primary key for our temp-tables in H2? I've seen some parts of FWD code using "custom" primary keys based on converted code, but I am not sure we actually convert these 4GL primary keys as primary keys inside H2. If the answer is no, do we always emit a (multiplex, recid) index, even if not primary?
  - I also have a bold idea for recid look-up. H2 uses internal unique keys (_ROWID_) for each table record, much like we do with recid. The look-up using the _ROWID_ is extremely fast, as it simply represents the index of the record inside the scan index. We can do a very slim recid to _ROWID_ hash mapping to benefit from this speed, but it we would still add a bit of overhead (on insert, delete and find by recid). AFAIK, the recid we use for temp-records are sequence generated and should just satisfy the uniqueness property. **Therefore**, can we consider that the temp-record recid is always equal to the internal H2 key? This way, the recid to _ROWID_ mapping is an identity mapping, so WHERE recid(tt) = 2 means the second record from the scan index.

#5 - 02/01/2023 10:54 AM - Alexandru Lungu

I've finished implementing a simple mean of H2 direct access for RAQ with simple recid look-up. The results are fairly good, in the sense that my custom test-cases work as expected and are usually faster with direct access. Sometimes, they are even with 60% faster than classical SQL access, but I also have tests with only 10% improvement. Regarding my last comment:

- HYDARTION: I simply wrap the rows into a dummy ResultSet to make use of FWD's compatibility to hydrate using ResultSet
- SELECT: I only apply direct access for tables with no computed column
- WHERE: I empirically discovered that (recid) is always the primary key of our temp-tables in H2; multiplex is irrelevant. This part can be extended to cover other unique indexes.

On a large customer application, I have the following statistic for RAQ.execute, extracted with JMX:

Scenario	Description	Count	Time (ms)	Avg. per query (ms)	% of the query count
FF Cache Hit	the record is fetched from the session cache	6400	252	0.03	15.95 %
FF	the	6108	0	0	15.22

Cache Hit (No Record)	cached result is FastFindCache.NO_RECORD				%
Direct Access	the dmo is manually found using the recid index	146	7	0.04	0.36 %
SQL Query	RAQ.executeImpl is used to retrieve the dmo	27465	2790	0.1	68.45 %

This shows that Direct Access is very insignificant as number of calls. This is because most of the temp-tables have a custom primary-key. After some debugging, I found out that there are lots of find queries (UNIQUE, FIRST, LAST or CAN-FIND) using the primary key fields in the where clause.

Good news is that the average time spent on direct access is very low comparing with SQL query and is mostly comparable with a ffcache look-up. Of course, this statement may be a bit off as only 146 queries use direct access and usually these are very simple. Best case scenario however is that direct access is actually 2 times faster, so most of that 27465 can cut their performance time in half.

Last thing to note is #7088 also affects direct access. My statistic doesn't include the time to generate the activeBundle (used only by SQL Query). Lazily computing the activeBundle may show a bigger gap between SQL and FFCache/Direct Access.

I am planning to extent my solution to also cover the cases in which all fields of an unique index (usually primary key) are used in the where clause with "=". My target is >30% of the find queries to use direct access.

#6 - 02/01/2023 12:54 PM - Eric Faulhaber

Alexandru Lungu wrote:

> I've finished implementing a simple mean of H2 direct access for RAQ with simple recid look-up. The results are fairly good, in the sense that my custom test-cases work as expected and are usually faster with direct access. Sometimes, they are even with 60% faster than classical SQL access, but I also have tests with only 10% improvement.

This is a nice improvement!

> Regarding my last comment:
>
> - HYDARTION: I simply wrap the rows into a dummy ResultSet to make use of FWD's compatibility to hydrate using ResultSet

Please also consider #6720; I don't yet know if lazy hydration should be applied to H2 embedded databases. I have to measure whether there is any significant time spent with hydration in this case. I suspect it will be less expensive than with a remote database (where there is some translation going on within the JDBC driver between the raw data coming over the wire and the Java types that need to be returned from ResultSet.getXXX() calls). So, it may not be worth enabling lazy hydration for H2 embedded databases.

> - SELECT: I only apply direct access for tables with no computed column

Is this necessary if the *table* has a computed column, or only if a computed column is involved in the query?

> - WHERE: I empirically discovered that (recid) is always the primary key of our temp-tables in H2; multiplex is irrelevant. This part can be extended to cover other unique indexes.

This is by design; there should never be more than one instance of a temp-table DMO with the same primary key (recid column), regardless of the multiplex ID. This is done so that we can use the same code to uniquely identify a record for temp-tables and for persistent tables. For example, we fetch from the session cache using a RecordIdentifier, which uses the primary key and not the multiplex ID.

One exception to this rule is if we have a copy of a DMO instance used for "housekeeping" (persistence internal) purposes, but that copy must never be used as the primary DMO (i.e., never stored in the session cache or any record buffer).

> On a large customer application, I have the following statistic for RAQ.execute, extracted with JMX:

Scenario	Description	Count	Time (ms)	Avg. per query (ms)	% of the query count
FF Cache Hit	the record is fetched from the session cache	6400	252	0.03	15.95%
FF Cache Hit (No Record)	the cached result is FastFindCache.NO_RECORD	6108	0	0	15.22%

Direct Access	the dmo is manually found using the recid index	146	7	0.04	0.36%
SQL Query	RAQ.executeImpl is used to retrieve the dmo	27465	2790	0.1	68.45%

> This shows that Direct Access is very insignificant as number of calls. This is because most of the temp-tables have a custom primary-key. After some debugging, I found out that there are lots of find queries (UNIQUE, FIRST, LAST or CAN-FIND) using the primary key fields in the where clause.

Please clarify what is meant by "the primary key fields" in this context. Do you mean the surrogate primary key (i.e., the column recid), or other fields of the temp-table that comprise unique indices defined in the 4GL code?

> Good news is that the average time spent on direct access is very low comparing with SQL query and is mostly comparable with a ffcache look-up. Of course, this statement may be a bit off as only 146 queries use direct access and usually these are very simple. Best case scenario however is that direct access is actually 2 times faster, so most of that 27465 can cut their performance time in half.

Please help me understand the intention here. Are you saying the idea is to apply this technique more generally, for some or all of the RAQ.executeImpl cases?

> Last thing to note is #7088 also affects direct access. My statistic doesn't include the time to generate the activeBundle (used only by SQL Query). Lazily computing the activeBundle may show a bigger gap between SQL and FFCache/Direct Access.
>
> I am planning to extent my solution to also cover the cases in which all fields of an unique index (usually primary key) are used in the where clause with "=". My target is >30% of the find queries to use direct access.

Is this the answer to my question above about the RAQ.executeImpl cases? Again, I am a bit confused by your description "all fields of an unique index (usually primary key)" in terms of what you mean by "primary key" in this context. Please clarify.

Nice work!

#7 - 02/01/2023 01:09 PM - Greg Shah

> I empirically discovered that (recid) is always the primary key of our temp-tables in H2; multiplex is irrelevant.

I don't understand this part. I admit I'm surprised that the same recid cannot be present with more than one of the multiplex id. That doesn't make sense to me since we should have cases in old 4GL code where we have to copy a temp-table that is passed in to a procedure or one that is copied back to the caller. In such cases I would have expected the recid to be duplicated and the multiplex id to be important.

> This shows that Direct Access is very insignificant as number of calls. This is because most of the temp-tables have a custom primary-key. After some debugging, I found out that there are lots of find queries (UNIQUE, FIRST, LAST or CAN-FIND) using the primary key fields in the where clause.

Does -rereadnolock (see #2175, #2176) factor in to this? I'm not sure if it is only for cross-session usage on permanent database tables.

#8 - 02/01/2023 01:15 PM - Eric Faulhaber

Alexandru, one other thing to consider, which I suppose you've already noticed: we don't actually create unique indices (except for the surrogate primary key index) in H2, when representing temp-table indices which were defined as unique in the legacy code. This is to allow us to flush records to the database "early" (i.e., before they pass unique constraint validation) without a SQL error, in order that they can be found by a query which expects them to be available. Unique constraints ultimately are enforced by the persistence validation code in the Validation class, rather than by H2.

#9 - 02/01/2023 01:25 PM - Eric Faulhaber

Greg Shah wrote:

> > I empirically discovered that (recid) is always the primary key of our temp-tables in H2; multiplex is irrelevant.
>
> I don't understand this part. I admit I'm surprised that the same recid cannot be present with more than one of the multiplex id. That doesn't make sense to me since we should have cases in old 4GL code where we have to copy a temp-table that is passed in to a procedure or one that is copied back to the caller. In such cases I would have expected the recid to be duplicated and the multiplex id to be important.

A copy in a separate logical or physical temp-table should have a distinct primary key (recid) value. This is the intent, at least. If our implementation differs, this could be a problem.

> > This shows that Direct Access is very insignificant as number of calls. This is because most of the temp-tables have a custom primary-key. After some debugging, I found out that there are lots of find queries (UNIQUE, FIRST, LAST or CAN-FIND) using the primary key fields in the where clause.

> Does -rereadnolock (see #2175, #2176) factor in to this? I'm not sure if it is only for cross-session usage on permanent database tables.

This should only apply to permanent tables where records can be read NO-LOCK and held in one session and become stale due to being changed simultaneously in another session. It seems this was meant to be a way to be "optimistic" (i.e., not holding a SHARE lock on a record). I think this is obsolete with our DMO versioning. Essentially, we always behave as if -rereadnolock was active.

#10 - 02/01/2023 01:26 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Alexandru, one other thing to consider, which I suppose you've already noticed: we don't actually create unique indices (except for the surrogate primary key index) in H2, when representing temp-table indices which were defined as unique in the legacy code.

To be clear: we create the index, it is just not marked unique.

#11 - 02/01/2023 01:29 PM - Greg Shah

> A copy in a separate logical or physical temp-table should have a distinct primary key (recid) value. This is the intent, at least. If our implementation differs, this could be a problem.

You're saying that in the 4GL these have separate recid values when a temp-table is passed as a parameter?

> I think this is obsolete with our DMO versioning. Essentially, we always behave as if -rereadnolock was active.

Can you explain this a bit? What do you mean by DMO versioning?

Considering that the default in OpenEdge is for this to be turned off, we may be missing an opportunity for optimization here. In fairness, most of the customer applications we have seen do use -rereadnolock but not all of them do.

#12 - 02/02/2023 05:51 AM - Alexandru Lungu

> Please also consider #6720; I don't yet know if lazy hydration should be applied to H2 embedded databases. I have to measure whether there is any significant time spent with hydration in this case. I suspect it will be less expensive than with a remote database (where there is some translation going on within the JDBC driver between the raw data coming over the wire and the Java types that need to be returned from ResultSet.getXXX() calls). So, it may not be worth enabling lazy hydration for H2 embedded databases.

My implementation uses SQLQuery.hydrateRecord to get the DMO based on the H2 internal values, so any lazy-hydration beyond hydrateRecord still holds. **However**, I always use the full row structure of the target DMO after H2 direct access, so all fields will be hydrated. Right now, lazy-hydration is detected when converting FQL to SQL, which is too late, as direct access happens before FQL generation.

> Is this necessary if the table has a computed column, or only if a computed column is involved in the query?

If the table has a computed column, we can't use direct access. I don't have a "select list" mechanism when doing direct access; I purely extract a reference to the searched row inside the primary key index. If the table has a computed column, then I would need to tell H2 to cherry-pick the values I am interested in: the ones used for hydration. Such "extra" features are to be avoided in direct access. Having too many will introduce overhead - one that we want to avoid when using SQL in the first place.

> Please clarify what is meant by "the primary key fields" in this context. Do you mean the surrogate primary key (i.e., the column recid), or other fields of the temp-table that comprise unique indices defined in the 4GL code?

Other fields of the temp-table that comprise unique indices defined in the 4GL code are used in the find where clause. I have many scenarios like:

```
define temp-table tt field myid as int index idx1 AS UNIQUE PRIMARY f1 ASCENDING.
[...]
find tt where myid = x. // the business logic doesn't rely on recid, but on a custom unique primary-key
[...]
```

I understand the confusion here. 4GL tables always have a primary key (implicit or explicit). By "custom" primary key, I meant an explicit unique primary index that has the same role as recid in the business logic. I've seen multiple times a key field that is indexed using an UNIQUE PRIMARY INDEX. Most of the FIND queries use where tt.key = ? instead of where recid(tt) = ?. Other examples include multi-field unique primary indexes (where tt.last-name=? and tt.first-name=?).

> Please help me understand the intention here. Are you saying the idea is to apply this technique more generally, for some or all of the RAQ.executeImpl cases?

"For some" rather than "for all". Note that the table also includes RAQ for persistent buffers, which are out of this task's scope.

I have the safest possible implementation right now and there are conditions which can be relaxed. I am implementing the following one by one until we have a relevant % for direct access.

- I only consider find queries in which the where clause is exactly tt.recid = ? (with or without query parameter). This can be extended for all "unique" indexes, not only recid.
- I only consider tables without computed-columns (date-time, extent or case-insensitive char fields). Once we have access into H2, we can do some magic to let H2 know which are the computed columns, so it can automatically do the cherry-picking I was mentioning above.
- I only target RAQ.execute with FIRST/LAST/UNIQUE keys. Technically, direct access can be also used directly in Loader.load or single-table Adaptive/Preselect queries.
- I always use direct acess only in single-table use-cases.
- Direct access can have other APIs (e.g. for FIND without WHERE, FIND over non-unique index).

> Alexandru, one other thing to consider, which I suppose you've already noticed: we don't actually create unique indices (except for the surrogate primary key index) in H2, when representing temp-table indices which were defined as unique in the legacy code. This is to allow us to flush records to the database "early" (i.e., before they pass unique constraint validation) without a SQL error, in order that they can be found by a query which expects them to be available. Unique constraints ultimately are enforced by the persistence validation code in the Validation class, rather than by H2.
> To be clear: we create the index, it is just not marked unique.

This information was latent somewhere in my mind. This is OK as I can pick the right index to do the look-up (even if not unique). However, I need a bit of feedback on the following scenario:

- there are two records with the same unique key inside the database (as one of them wasn't validated yet by FWD).
- I execute something like FIND tt WHERE tt.key = ?
- At this point, is there a resolution of which of the two records with the same unique key is retrieved. Or the FIND statement will trigger validation, so the database look-up won't be done anyways?
- If we use some resolution to choose between the two records, the direct access needs to be aware of that, so it can replicate.

  > You're saying that in the 4GL these have separate recid values when a temp-table is passed as a parameter?

Just tested in 4GL with two procedures:

- First one creates a record that has id 2304. It uses its temp-table as an **input-output** parameter for the second procedure, where the record now has id 4352. A new record in the second procedure is assigned 4353. Returning into the first procedure I have 2305 and 2306.
- The conclusion is that recid is reassigned when used as parameter (in a multiplexed table in our case).
- It is not clear if the set of ids from the first table can ever overlap the set of ids from the second table. However, it is clear that they are not copied, but reassigned. Therefore, this is not a feature one customer can ever rely on IMO.

  > Does -rereadnolock (see #2175, #2176) factor in to this? I'm not sure if it is only for cross-session usage on permanent database tables.

I need to do some reading first. On the surface, it looks like a permanent database setting, but I will do some digging for the temp-table case.

#13 - 02/02/2023 09:54 AM - Greg Shah

> I only consider tables without computed-columns (date-time, extent or case-insensitive char fields)

In regard to extents, we are moving to "expanded" mode for all cases in the future. Does that open up these tables for direct usage?

In regard to case-insensitive char fields, nearly 100% of char fields in the 4GL are treated case-insensitively. Does this mean we are excluding any table that has even one field that is just a normal case-insensitive character type? That seems like just about every table would have such a thing.

#14 - 02/02/2023 11:46 AM - Alexandru Lungu

Greg Shah wrote:

> > I only consider tables without computed-columns (date-time, extent or case-insensitive char fields)
>
> In regard to extents, we are moving to "expanded" mode for all cases in the future. Does that open up these tables for direct usage?

I am aware of that and I think this is close to acceptable for direct access. I need to do a test to ensure this works; the main requirement is to have all extents in ascending order as a contiguous segment, which I think is already happening.

> In regard to case-insensitive char fields, nearly 100% of char fields in the 4GL are treated case-insensitively. Does this mean we are excluding any table that has even one field that is just a normal case-insensitive character type? That seems like just about every table would have such a thing.

I may have misinterpreted FWD's behavior. I have seen multiple fields like "some_field" and "isome_field" in the database; basically each character field is doubled: a native one and an upper-case one. This affects direct access, because it can't make a distinction between the native "some_field" and the computed "isome_field" yet. However, this only holds for **indexed** character fields and not case-insensitive ones; I need to double-check.

Anyways, this is not fatal for direct access. If there is a low rate exactly because of computed columns, I can easily mark these columns, such that the direct access can identify them.

#15 - 02/02/2023 11:56 AM - Eric Faulhaber

Alexandru Lungu wrote:

> I may have misinterpreted FWD's behavior. I have seen multiple fields like "some_field" and "isome_field" in the database; basically each character field is doubled: a native one and an upper-case one. This affects direct access, because it can't make a distinction between the native "some_field" and the computed "isome_field" yet. However, this only holds for **indexed** character fields and not case-insensitive ones; I need to double-check.

Yes, the computed columns exist for those character columns which we need to index (whether case-sensitive or case-insensitive) in a 4GL-compatible way (i.e., upper/rtrim'd for case-insensitive fields, or just rtrim'd for case-sensitive fields). The computed columns exist only because H2 (like Maria and SQL Server and unlike PostgreSQL) does not support expressions/functions in index definitions, so the computed columns are defined with those functions applied, and then referenced in indices. If a field is not indexed, it will not have an associated, computed column.

#16 - 02/02/2023 03:47 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> You're saying that in the 4GL these have separate recid values when a temp-table is passed as a parameter?

It happened that I was working on fixing a flaw of my implementation of the expanded extent fields (related to fast-copy) and I tested this. It turned out that in 4GL the answer is yes and no. The records will keep the recids when invoking an internal procedure and will have new (intermediary) recids when invoking an external procedure. I get an output like this:

```
caller1:   10496 9 yes no ? master
caller2:   10497 -9 ? yes no master
internal1: 10496 9 yes no ? master
internal2: 10497 -9 ? yes no master
external1: 8448 9 yes no ? master
external2: 8449 -9 ? yes no master
```

The caller creates two records and calls two identical procedures, one internal and one external.

I guess FWD does the same, but since the recids restart at 1, it is difficult to say for sure without further investigations.

Alexandru Lungu wrote:

> Greg Shah wrote:
>
>> In regard to extents, we are moving to "expanded" mode for all cases in the future. Does that open up these tables for direct usage?
>
> I am aware of that and I think this is close to acceptable for direct access. I need to do a test to ensure this works; the main requirement is to have all extents in ascending order as a contiguous segment, which I think is already happening.

Yes, the expanded properties are 'inlined' replacing the original property. For example:

```
define temp-table tt1 field f1 as date field f2 as int extent 5 field f3 as char
```

will be use a SQL table created as:

```
create table tt1 (recid bigint not null, f1 date, f2_1 integer, f2_2 integer, f2_3 integer, f2_4 integer, f2_5
 integer, f3 varchar)
```

#17 - 02/03/2023 10:34 AM - Alexandru Lungu

I extended my solution to cover all unique indexes. Most of the processing is done in FQLPreprocessor to identify where clauses like tt.tenant = ? and tt.key = 2 (I will call them "unique finds"). If all clauses use = and cover strictly all fields of an unique index, then direct access is used. I profiled the overhead of collectPropertyMatches (43ms) and checkUniqueFind (23ms). The total time is low considering that only 4589 preprocessors are actually created (for all ~40.00 find queries). Please note that I enabled collectPropertyMatches for all queries (not only informational). Please consider that such preprocessing step can help other FWD parts as well (much like isFindByRowid does).

I've redone my test with the new preprocessing step.

Scenario	Description	Count	Time (ms)	Avg. per query (ms)	% of the query count
Direct Access (recid)	the record is retrieved from H2	146	7	0.04	0.36%
Direct Access ("unique finds")	the record is retrieved from H2	56	2	0.03	0.14%

Again, the count is not considerable. The new additions (56) are simply some uniquely indexed handle fields which are rarely accessed. I added some JMX and out of that ~27,000 find queries using SQL, ~12,500 are in fact "unique finds". However, the condition that doesn't allow these ~12,500 to use direct access is the "can't use direct access with tables that have computed columns" constraint. Out of these, ~7,000 have indexed character columns. I am planning to target this issue next.

> Yes, the computed columns exist for those character columns which we need to index (whether case-sensitive or case-insensitive) in a 4GL-compatible way (i.e., upper/rtrim'd for case-insensitive fields, or just rtrim'd for case-sensitive fields). The computed columns exist only because H2 (like Maria and SQL Server and unlike PostgreSQL) does not support expressions/functions in index definitions, so the computed columns are defined with those functions applied, and then referenced in indices. If a field is not indexed, it will not have an associated, computed column.

My concern right now isn't that we have an extra column to handle this use-case, but the fact that I need to remove it "manually" from the values array before returning to FWD. Further, this is not a performance issue, but I can't identify these computed columns (checking that they start with "i" is far from safe). From this point:

- we can do some H2 internal changes to adapt to our use-case: implement expression/function in index definitions (not trivial) or force all character indexed fields inside H2 to be compared using upper-case. Please let me know if we can go ahead with any of these two. I prefer the second one due to less overhead and easier/cleaner implementation. If so, my next suggestion won't be needed.
- also useful for #7060, consider a COMPUTED or HIDDEN keyword for table columns which are not meant to be retrieved using direct access or, in the case of #7060, with wildcard.

#18 - 02/03/2023 04:20 PM - Ovidiu Maxiniuc

Actually the prefixes for the computed columns in H2 dialect are __i and __s (for **i** gnore casing and case- **s** ensitive data character types). We already use P2JH2Dialect.extractColumnName() to extract back the original column name from an index metadata.

I agree that usually it's a risky method to check the prefix this way but, normally, there should be no collisions. The double underscore prefix was intentionally selected for this reason.

As you noted, these computed columns do not need to be fetched from the SQL. They are only used as part of the index to help sorting the (null) values the same way as in 4GL.

#19 - 02/03/2023 05:17 PM - Eric Faulhaber

Ultimately, we create the SQL forms of the column names. So, we can use whatever we want as an identifier, if we find the prefixes __i or __s are not distinctive enough (though I think they are, as I don't think we ever create "normal" columns with leading underscores for temp-tables, do we?).

> They are only used as part of the index to help sorting the (null) values the same way as in 4GL.

I assume they are also used to "bake in" the [upper]/rtrim as well, correct?

#20 - 02/03/2023 07:32 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ultimately, we create the SQL forms of the column names. So, we can use whatever we want as an identifier, if we find the prefixes __i or __s are not distinctive enough (though I think they are, as I don't think we ever create "normal" columns with leading underscores for temp-tables, do we?).

No, we do not.
In the absence of a SQL option to SELECT * EXCEPT __ifield1, __sfield2 FROM... I think we can analyse the ratio of computed columns before sending the query to execute and, if there are only a few or none indexed strings we can use the * and just ignore these additional fields when the record is rehydrated. Otherwise we risk doubling the amount of data fetched from SQL so we probably better to explicitly specify each column in the query as we do now.

> > They are only used as part of the index to help sorting the (null) values the same way as in 4GL.
>
> I assume they are also used to "bake in" the [upper]/rtrim as well, correct?

Yes, sorry, my mistake. When the DDLs are constructed and everywhere a specific index is required (sorting phrases). The order of null values is dictated by nulls last on definition of each index component.

#21 - 02/06/2023 04:43 AM - Alexandru Lungu

> Actually the prefixes for the computed columns in H2 dialect are __i and __s (for i gnore casing and case- s ensitive data character types). We already use P2JH2Dialect.extractColumnName() to extract back the original column name from an index metadata.

Oh right! I misinterpreted the CC_PREFIX addition.

> In the absence of a SQL option to SELECT * EXCEPT __ifield1, __sfield2 FROM... I think we can analyse the ratio of computed columns before sending the query to execute and, if there are only a few or none indexed strings we can use the * and just ignore these additional fields when the record is rehydrated. Otherwise we risk doubling the amount of data fetched from SQL so we probably better to explicitly specify each column in the query as we do now.

This makes sense; the weak spots of #7060 are lazy-hydration and the computed columns. Once we decide to filter-out these computed columns in FWD (for low-ratio), lazy-hydration is the last to trouble #7060.

> force all character indexed fields inside H2 to be compared using upper-case. Please let me know if we can go ahead with any of these two.

What is you input on this? On my second thought, this can cover more use-cases, not only indexed char fields.

We can use something similar to the native type from PostgreSQL (as in #6837, citext) as a native data type in FWD-H2. In fact, H2 already has VARCHAR_IGNORECASE and VARCHAR_CASESENSITIVE as native types. Even if they may not fully fit our FWD use-cases out-of-the-box, we can do some tuning. If RTRIM is something we worry about, we can augment the field definition with RTRIM keyword (column RTRIM VARCHAR_IGNORECASE).

Anyways, we should agree on a single way of doing this. Checking __i and __s are going to be irrelevant for temp-dabase, if we use VARCHAR_IGNORECASE. The same applies for #7060 using *. If we plan to go with VARCHAR_IGNORECASE, I will do a separate issue for it and start working on it, before moving on with #7062.

#22 - 02/09/2023 10:42 AM - Alexandru Lungu

Alexandru Lungu wrote:

> > In the absence of a SQL option to SELECT * EXCEPT __ifield1, __sfield2 FROM... I think we can analyse the ratio of computed columns before sending the query to execute and, if there are only a few or none indexed strings we can use the * and just ignore these additional fields when the record is rehydrated. Otherwise we risk doubling the amount of data fetched from SQL so we probably better to explicitly specify each column in the query as we do now.
>
> This makes sense; the weak spots of #7060 are lazy-hydration and the computed columns. Once we decide to filter-out these computed columns in FWD (for low-ratio), lazy-hydration is the last to trouble #7060.

Just found out that H2 supports INVISIBLE keyword all along (for the exact purpose of #7060). Now, I generate it for computed columns in H2 and use it to filter-out computed columns for direct access. It works just fine. I am moving on with this issue.

> > force all character indexed fields inside H2 to be compared using upper-case. Please let me know if we can go ahead with any of these two.
>
> What is you input on this? On my second thought, this can cover more use-cases, not only indexed char fields.
>
> We can use something similar to the native type from PostgreSQL (as in #6837, citext) as a native data type in FWD-H2. In fact, H2 already has VARCHAR_IGNORECASE and VARCHAR_CASESENSITIVE as native types. Even if they may not fully fit our FWD use-cases out-of-the-box, we can do some tuning. If RTRIM is something we worry about, we can augment the field definition with RTRIM keyword (column RTRIM VARCHAR_IGNORECASE).

I created #7108 for further investigation. This is in close relation with #6837, but dedicated to H2.

#23 - 02/10/2023 08:22 AM - Alexandru Lungu

- % Done changed from 10 to 70

I've redone the tests with direct access:

- Computed columns are now marked as invisible in H2.
- Before returning from the H2 embedded FWD driver, the invisible columns are removed.
- Direct access is not yet applied for tables with extents. Testing this now!

Scenario	Description	Count	Time (ms)	Avg. per query (ms)	% of the query count
FF Cache Hit	the record is fetched from the session cache	7273	247	0.03	17.75%
FF Cache Hit (No Record)	the cached result is FastFindCache.NO_RECORD	6777	0	0	16.53%
Direct Access (Unique)	the dmo is manually found using an unique index	12718	143	0.01	31.03%
Direct Access	the dmo is manually found using the recid index	436	3	0.006	1.06%
SQL Query	RAQ.executeImpl is used to retrieve the dmo	13770	1601	0.11	33.60%

This test is based on the same POC as the one in #7062-5. The number of direct access find queries increased drastically and the avg. time per query is way lower.
I will test how direct access behaves with extent fields. Afterwards, I am planning to create a 7062_h2 branch to commit these changes for review.
Fortunately, I will be able to integrate this change into #7058 track in time.

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

- % Done changed from 70 to 90

- File direct_access.patch added

I committed 7062a_h2/rev. 9 the changes to H2 to allow direct access.
There is a patch here on the changes required by FWD. Note that the changes are highly dependent on having a built version of H2 using 7062_h2/rev. 9, as it uses internal structures from H2. Please review.
I am planning to make some final changes to 7062a (javadoc and small performance changes), before merging.

#25 - 02/15/2023 06:53 AM - Alexandru Lungu

Alexandru Lungu wrote:

> I am planning to make some final changes to 7062a (javadoc and small performance changes), before merging.

Rebased 7062a_h2, committed javadoc and some minor changes. I am doing a quick final profiling step right now before merging into FWD-H2 trunk.

#26 - 02/16/2023 09:07 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 90 to 100

Merged 7062a_h2 into trunk after adding support for some extra data-types from FWD (object as resourceId and decimal as BigDecimal). FWD-H2 trunk is now at rev. 11.
Created 7062a branch for FWD changes and committed rev. 14485 including an updated version of the #7062-24 patch.

#27 - 02/24/2023 06:58 AM - Alexandru Lungu

I merged the changes from 7062a into 7026a - the branch names are quite confusing :)

I have good results, similar to what I expected from #7062-23 in total.

However, I've done some debugging to identify where we still spend time in direct-access. Quite surprising, some calls to DirectAccessHelper.hydrateFromResultSet were hitting the session cache, so sometimes the whole direct-access code is used only to retrieve the recid to be used as a cache key. This was still happening before direct-access, because FWD wasn't aware of the recid before executing the SQL. The usage of the cache was just to bypass hydration, even if the full-record was already retrieved from H2. Please note that we don't do projection queries for temp-tables.

Now that we have direct access, we can return the recid and a supplier function to obtain the result-set if required. Therefore, we avoid building the result-set if the session cache is hit. If the record is not in cache, we can call the supplier and hydrate from result-set. I don't expect a **huge** improvement, because direct-access is no longer copying values from internal H2 structures to result-set anyways. On the other hand, this way we can avoid a for loop of all columns (usually tens, but with extent denormalization, this can mean hundreds).

I will need to check how many times such scenario is encountered.

#28 - 02/27/2023 07:47 AM - Constantin Asofiei

Alexandru, the changes in RandomAccessQuery will not set the lock properly on a record found in the cache:

```
+          Record dmo = null;
+
+          if (lockType == null)
+          {
```

```
+            lockType = this.lockType;
+         }
+
+         if (fqlPreprocessor.isFindByRowid())
+         {
+            Session session = buffer.getSession(true);
+            try
+            {
+               dmo = session.getCached(buffer.getDmoInfo().getImplementationClass(), fqlPreprocessor.getFindB
yRowid(values));
+               if (dmo != null)
+               {
+                  finalizeFind(lockType, dmo, () -> buffer.errorNotOnFile(), true);
+               }
+            }
+            catch (PersistenceException e)
+            {
+               // fall-back to execute method
+            }
+         }
+
+         if (dmo == null)
+         {
+            activateUnique();
+            dmo = execute(values, lockType, true, true);
+         }
+
```

finalizeFind is not calling persistence.lock, it just updates the internal RecordBuffer.pinnedLocks. I think the correct code for a cached record should be this:

```
               if (dmo != null)
               {
                  // if we get a hit, get the requested lock with RecordLockContext
                  RecordLockContext lockCtx = buffer.getRecordLockContext();
                  lockCtx.lock(new RecordIdentifier<>(buffer.getTable(), dmo.primaryKey()), lockType);
               }
```

Also, original version is calling finalizeFind twice.

#29 - 02/27/2023 07:58 AM - Alexandru Lungu

This was a last minute modification and the LOCK_MODE 0 from H2 was stuck in my head. It struck me at some point that this can work for persistent databases too and I let it be this way for any arbitrary RAQ.
Your suggestion from #7062 fits - is the same procedure ffCache uses, so I guess is the right way to do it. Please let me know if I can commit this change to 7026a?

#30 - 02/27/2023 08:01 AM - Alexandru Lungu

```
=== modified file 'src/com/goldencode/p2j/persist/RandomAccessQuery.java'
--- src/com/goldencode/p2j/persist/RandomAccessQuery.java    2023-02-24 12:36:40 +0000
+++ src/com/goldencode/p2j/persist/RandomAccessQuery.java    2023-02-27 13:00:02 +0000
@@ -2326,15 +2326,17 @@
             lockType = this.lockType;
          }

-         if (fqlPreprocessor.isFindByRowid())
+         if (fqlPreprocessor.isFindByRowid() && whereExpr == null && isSimpleQuery())
          {
             Session session = buffer.getSession(true);
             try
             {
-               dmo = session.getCached(buffer.getDmoInfo().getImplementationClass(), fqlPreprocessor.getFindB
yRowid(values));
+               Class<? extends Record> dmoClass = buffer.getDmoInfo().getImplementationClass();
+               dmo = session.getCached(dmoClass, fqlPreprocessor.getFindByRowid(values));
                if (dmo != null)
                {
-                  finalizeFind(lockType, dmo, () -> buffer.errorNotOnFile(), true);
+                  RecordLockContext lockCtx = buffer.getRecordLockContext();
+                  lockCtx.lock(new RecordIdentifier<>(buffer.getTable(), dmo.primaryKey()), lockType);
                }
             }
             catch (PersistenceException e)
```

I will also allow such optimization for queries without client-where, joins and external buffers - keep it simple.

#31 - 02/27/2023 08:04 AM - Constantin Asofiei

Alexandru Lungu wrote:

> [...]
> I will also allow such optimization for queries without client-where, joins and external buffers - keep it simple.

I think it looks OK, please commit it.

Please do a more thorough review of the changes in 7026a revs. 14500, 14501, 14503, 14504, to cleanup the code (at least for RandomAccessQuery class fields are defined out-of-place, they are missing javadoc, etc).

#32 - 02/27/2023 09:24 AM - Alexandru Lungu

Done, committed 7026a/rev. 14506.

#33 - 02/28/2023 02:49 AM - Alexandru Lungu

Committed 7026a/14507 removing some tracing variables - missed them on my first check.

#34 - 03/07/2023 08:44 AM - Alexandru Lungu

I've done the performance test on another large customer application. I've added more details to cover temp vs persistent performance + direct access vs SQL on the exact same queries. I have the same scale as the results from #7062-5 and a slight worse performance than #7062-23, but still direct-access shows a 3 times improvement. The tests include #7143-38 fix.

Scenario	Description	Count	Time (ms)	Avg. per query (ms)	% of the query count
FF Cache Hit	the record is fetched from the session cache	1978	50	0.025	20.9%
FF Cache Hit (No Record)	the cached result is FastFindCache.NO_RECORD	214	0	0	2.26%
Direct Access *	the dmo is manually found using an unique index	2638	76	0.028	27.9% *
SQL Query on Unique *	the dmo is found using SQL, but could have used direct-access	2638	230	0.087	27.9% *
Other SQL Query on Temporary	the temp dmo is found using SQL	3077	238	0.077	32.62%
Other SQL Query on Persistent	the persistent dmo is found using SQL	1523	1010	0.663	16.15%

The results marked with * are done on the same queries; RAQ.executeImpl done both direct-access and SQL query independently. "Other" SQL queries are the one which don't satisfy the direct-access requirements.
I think it is an important result showing that two different large applications have ~30% of their RAQ use direct-access, which consistently runs at least 3 times faster.

#35 - 03/07/2023 09:10 AM - Greg Shah

Do these results differ from 7026a rev 14503?

#36 - 03/07/2023 09:28 AM - Alexandru Lungu

The results were obtained using 7026a/rev. 14518. I don't have some results for 7026a/rev. 14503 for this specific application. [#7062-23]() were done with a very close version as rev. 14503, but were on a different application.

Now, 7026a was rebased, so the former rev. 14503 is the new rev. 14510. Therefore, it is hard to have an exact profiling of the former 7026a/rev. 14503 without accounting the rebased changes.

I don't expect a different result with rev. 14510 comparing to rev. 14518 anyway. The only related change is rev. 14518, which honored record nursery. The record nursery should be accounted both in direct-access and SQL queries. Note that I extracted the record nursery outside the profilers from [#7062-34](), to get only the execution times of the queries alone.

I've profiled the record nursery code right now, in case of direct-access, and it takes 12ms in that 2638 calls.

#37 - 03/14/2023 08:03 AM - Alexandru Lungu

Moved the isFindByRowid short-circuit from [#7062-30]() to RAQ.execute. This is because isSimpleQuery was always returning false, due to the fact that there was no active bundle. This way, the optimization is truly triggered and it covers more scenarios (including FIRST and LAST). I've worked with some MBeans and 100% (on a large customer project with ~500 of such hits) of the fqlPreprocessor.isFindByRowid() && whereExpr == null && isSimpleQuery() cases are actually finding the dmo is the session cache. Committed to 7026b/rev. 14503.

#38 - 03/16/2023 11:31 AM - Alexandru Lungu

Response to #7026-97

Constantin Asofiei wrote:

> Alexandru, I'm not sure where to post this. The changes for DirectAccessDriver can end up with a ResultSet instance with a NULL ResultSet.getStatement(). This getStatement() is being used in a few places in FWD, one of them being QueryProfilerJMX.updateCacheHits. You will see this NPE if you try to enable the JMX beans in e.g. VisualVM and run the performance tests.

Makes sense, as QueryProfilerJMX is grouping the queries with the same SQL. The issue is that a direct-access query is not using an SQL! I wasn't very fond of a dummy JDBCResultSet to deliver results from direct-access in the first place; maybe the best way was to have a more general hydration technique, which is based on a FWD results provider, not explicitly on java.sql.ResultSet. But at this point, the DataHandler logic is already too entangled with our hydration.

Fix 1: check if a statement exists. If it doesn't presume empty string is the "SQL" of a direct-access.
Fix 2: create a stub JDBCStatement for the result-set returned by direct-access. This way, we can build an SQL when required (lazily), based on the direct-access API called.

> How do we fix this? Please note that there are cases of Connection conn = rs.getStatement().getConnection(); for BlobType.readProperty() and ClobType.readProperty(), with the rest being used like ResultSet.getStatement().toString(), from QueryProfilerJMX.

For BlobType and ClobType, the statement is required to retrieve the connection and modify the auto-commit.

The only fix I see here: use a stub JDBCStatement for the result-set. Allow it to return the connection if needed.

#39 - 03/16/2023 11:50 AM - Constantin Asofiei

Go with option 2 - on getStatement() call, lazily create the java.sql.Statement instance which has all the entire JDBC state (not just query string or JDBC Connection). I don't want to break the JDBC semantics for ResultSet.getStatement().

#40 - 03/17/2023 08:48 AM - Alexandru Lungu

Created 7062b_h2 for further work on H2.

Committed 7062b_h2/rev. 13 - generating a dummy jdbc statement for each result-set of direct-access. The generation of one statement is not that costly. H2 itself doesn't do much when generating a simple statement; it hooks it with a connection and that is all. Unless it is prepared, H2 doesn't store the sql inside the statement. JdbcStatement.toString returns by default the id of the statement.

#41 - 03/20/2023 09:54 AM - Alexandru Lungu

I also committed an optimization for #7062-27 (7062b_h2/rev. 14 and 7026b/rev. 14512), including a mean of lazily computing the result-set. If the provided record is cached (by recid), avoid computing the result set from H2. This avoids extracting the data from the tables and place them into a separate array to be served by the result-set. This shows a -0.2% improvement.

#42 - 04/20/2023 07:12 AM - Alexandru Lungu

- Status changed from Review to Test

This was merged in trunk as rev. 14523 and can be closed.

#43 - 05/29/2023 08:41 AM - Constantin Asofiei

Alexandru, we need NPE protection here:

```
java.lang.NullPointerException
        at org.h2.embedded.FWDDirectAccessDriver.getUniqueRow(FWDDirectAccessDriver.java:131)
        at com.goldencode.p2j.persist.orm.DirectAccessHelper.findByUniqueIndex(DirectAccessHelper.java:152)
        at com.goldencode.p2j.persist.RandomAccessQuery.executeDirectAccess(RandomAccessQuery.java:4908)
        at com.goldencode.p2j.persist.RandomAccessQuery.execute(RandomAccessQuery.java:3610)
        at com.goldencode.p2j.persist.RandomAccessQuery.first(RandomAccessQuery.java:1571)
        at com.goldencode.p2j.persist.RandomAccessQuery.first(RandomAccessQuery.java:1456)
```

This is a case where the table does not exist. The same for the properties. Throw a SQLException in such cases.

#44 - 05/29/2023 09:40 AM - Alexandru Lungu

Created 7062c_h2 and fixed the NPE.
I didn't expect to have a direct-access on a table with invalid name. This is something that should be tracked back to FWD. Do you have some back-trace on this error? Maybe a weird name used?

Please let me know how fast do you want this integrated.

#45 - 05/29/2023 11:58 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Created 7062c_h2 and fixed the NPE.
> I didn't expect to have a direct-access on a table with invalid name. This is something that should be tracked back to FWD. Do you have some back-trace on this error? Maybe a weird name used?
>
> Please let me know how fast do you want this integrated.

What I'm seeing in some customer logs is most likely a side-effect of #7349, where the H2 session gets lost - the point was more to avoid NPEs and allow SQLException. There is a very small chance that the NPE is the root cause.

#46 - 06/09/2023 03:31 AM - Alexandru Lungu

Merged 7062c_h2 into FWD-H2 trunk as rev. 21.
The fix will be available with the next FWD-H2 upgrade.

Files

direct_access.patch	74.9 KB	02/10/2023	Alexandru Lungu