Database - Feature #7379

cache the def and bound DMO property info in TemporaryBuffer$ReferenceProxy

05/23/2023 10:31 AM - Constantin Asofiei

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

Description

History

#1 - 05/23/2023 10:40 AM - Constantin Asofiei

Created task branch 7379a from trunk rev 14584.

Rev 14585 adds a cache for the DMO property info required by TemporaryBuffer$ReferenceProxy. Ovidiu, should we have these in a different place? Please review.

#2 - 05/23/2023 03:40 PM - Ovidiu Maxiniuc

I think we already have (at least the BOUND_PROPERTIES). The method to call is defBuffer/boundBuffer.dmoInfo.getFields(false). It will return an iterator over the original fields from temp-table, in the order they were defined. Iterate them synchronously to create the property pair you need. If the parameter of getFields is true it will also include the 'reserved' properties which you extract as defaultProps.

Accessing the TableMapper is not recommended. It has a slow complex structure.

The PROPERTIES are probably in the best location.

#3 - 05/23/2023 03:47 PM - Constantin Asofiei

Thanks, Ovidiu. I just noticed that TableMapper.getAllLegacyFieldInfo is still unconditionally executed.

Is dmoInfo.getFields(false) using the same order as getAllLegacyFieldInfo?

#4 - 05/23/2023 03:51 PM - Constantin Asofiei

And there is also propertyIndexes used in ReferenceProxy.bind, which can be cached.

#5 - 05/23/2023 06:08 PM - Ovidiu Maxiniuc

- File buffer-binding2.png added

Constantin Asofiei wrote:

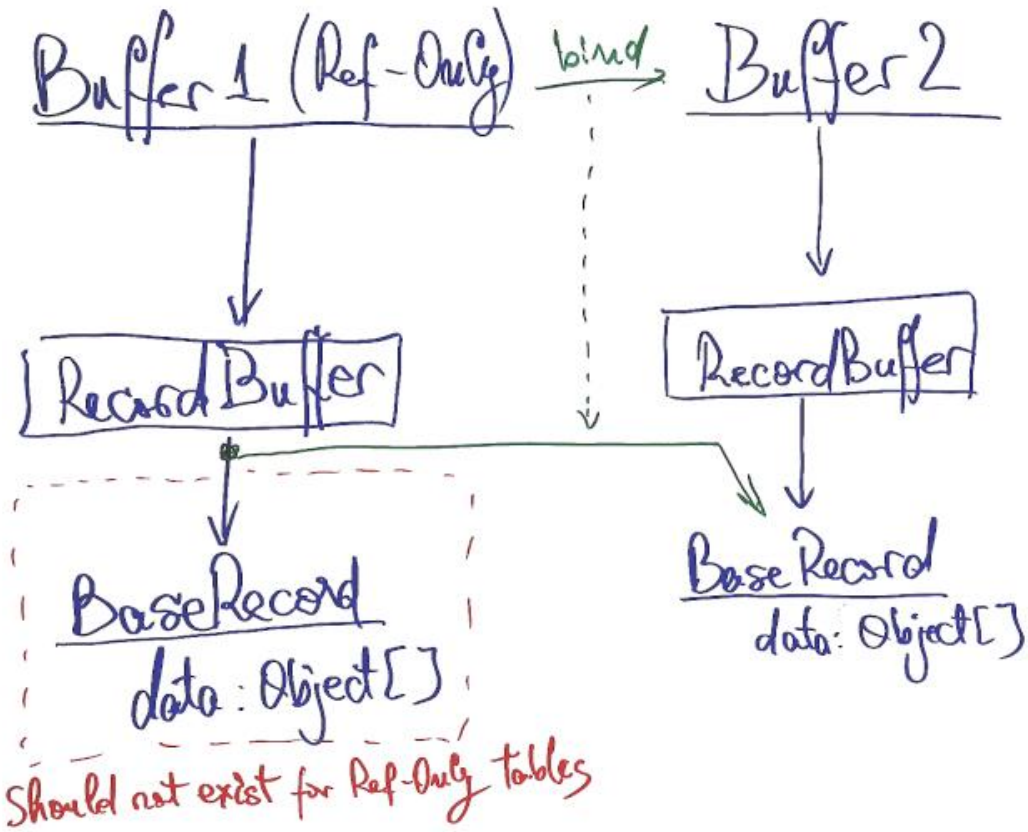
> Is dmoInfo.getFields(false) using the same order as getAllLegacyFieldInfo?

Actually no. The sorting happens on slightly different keys. The former use the propId, the latter use the id attribute. If the table is not expended (normalized) these values are identical. But when an extent field is expanded each of its properties receives a unique propId while they all keep the same id.

> And there is also propertyIndexes used in ReferenceProxy.bind, which can be cached.

propertyIndexes are the associated values of id.

However, I think the optimal solution here is to do the binding at BaseRecord level, by redirecting the accesses to data. This means **no mapping** (using Map) at all. And I am almost sure, **no proxies**, too. Here is a diagram sketch:



So we have the lowest level object, the data array of similar size and corresponding properties types (validated while binding). All we need to do is redirect R/W accesses of buffer1.currentRecord.data[i] to buffer2.currentRecord.data[i]. The Record/BaseRecord have to be kept since the access will still be done via the DMO interface, but the data it writes to/read from is temporarily (during bind) changed (maybe the lower green arrow should be moved even lower).

#6 - 05/24/2023 08:26 AM - Constantin Asofiei

Ovidiu, the binding maps is not only for accessing the record datum. This info is used when translating queries, too.

Also, binding is not limited to REFERENCE-ONLY tables. We can't remove ReferenceProxy support - see this test:

```
def temp-table tt1 field f1 as int.
def temp-table tt2 field f2 as int.

create tt1.
tt1.f1 = 10.
procedure proc1.
   def input parameter table for tt2 bind.

   find first tt2.
   message tt2.f2 string(temp-table tt2:handle) string(temp-table tt1:handle).  // this shows the same handle
value for both tt1 and tt2.
end.

run proc1(input table tt1 by-reference).
```

#7 - 05/24/2023 09:35 AM - Ovidiu Maxiniuc

My solution will not require translation at all: the buffer1 from my diagram will access the data of buffer2 using its own name and interface.

I did not say the solution is limited to REFERENCE-ONLY tables, just that they should have no RecordBuffer before being bound.

I have no solution for identical table handle yet :(.

#8 - 05/24/2023 09:39 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> My solution will not require translation at all: the buffer1 from my diagram will access the data of buffer2 using its own name and interface.

find first tt2 where tt2.f2 = 10 needs to translate and use the tt1.f1 table and field, at the SQL level. You don't have the record yet at that point. This is the translation I mean.

> I have no solution for identical table handle yet :(.

Yes, this is another reason for this complexity.

#9 - 05/24/2023 11:22 AM - Eric Faulhaber

Please note that directly accessing the BaseRecord.data buffer is going to go badly in cases of the FIELDS clause and lazy hydration (#6720). The latter (if I can make it performant) will be the default behavior for non-temp-tables. When I originally implemented that array, I envisioned "fast" scenarios, where the data array could be used more liberally within the persistence framework. However, I had not contemplated FIELDS clause and lazy hydration implementations back then.

Now, null elements in the data array do not necessarily mean "unknown value". They might mean uninitialized value instead. A separate bit set determines which meaning is valid. Bottom line is that directly doing R/W operations on the data array is no longer supported; we have to go through BaseRecord.{get|set}{Datum|Data} APIs to be safe. I am still working through the implementation of those for lazy hydration.

#10 - 05/24/2023 12:02 PM - Constantin Asofiei

Eric, I'm using OrmUtils.setField and OrmUtils.getField, not direct data array access. Is this of concern?

About FIELDS: I don't see how this can be used with a FILL query.

#11 - 05/24/2023 12:20 PM - Constantin Asofiei

7379a rev 14586 adds cache for propertyIndexes. TableMapper now is called only when there is a cache miss; so, even if it's expensive, it is called only once.

#12 - 05/24/2023 12:54 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric, I'm using OrmUtils.setField and OrmUtils.getField, not direct data array access. Is this of concern?

Hm, yes, but maybe more for me on the lazy hydration end. My work is primarily on the getter side. If a data element is null upon invoking BaseRecord.getDatum(int) (which OrmUtils.getField does), a null will be returned. This currently indicates that field has unknown value assigned.

However, with lazy hydration, for a DMO which was created from a query's result set, it may mean that the field's value has not yet been assigned. This information is managed in a separate bit set in BaseRecord. In that case, we will try to just-in-time hydrate the data element, if the result set is still valid, and we are still positioned on the correct row in it. If not, null is returned. null can still be returned if the field value is actually unknown value. So, a null return requires a follow-up call to check the bit set and determine the meaning.

I do it this way to avoid having to throw and catch an exception, which causes two problems:

- it is expensive; and
- unless it is unchecked (which means it cannot be a PersistenceException subclass), it would require re-working the DMO interfaces to declare a throws, which I want to avoid.

However, this means all callers of BaseRecord.getDatum have to be able to interpret a null properly. If it means just-in-time hydration failed, we must hydrate the field (and currently, all uninitialized fields) from a new query. I'm still working this part out, but I'm avoiding doing this within BaseRecord.getDatum directly, because it carries a lot more baggage in terms of possible error/exception handling, which we already deal with at higher levels.

If I have to deal with all this within OrmUtils, it is not ideal, but there may not be a better way.

> About FIELDS: I don't see how this can be used with a FILL query.

I was basing my comments generally on Ovidiu's diagram, so they aren't necessarily applicable in all cases.

#13 - 05/24/2023 01:20 PM - Eric Faulhaber

Constantin, the only place I see OrmUtils.getField used currently is in 3 locations in RecordBuffer.copy. Are you expanding this use?

#14 - 05/24/2023 01:23 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, the only place I see OrmUtils.getField used currently is in 3 locations in RecordBuffer.copy. Are you expanding this use?

Yes, via this in RecordBuffer:

```
   protected Object getDirectField(int offset)
   {
      return OrmUtils.getField(currentRecord, offset);
   }
```

called from BufferImpl.assign.

#15 - 05/24/2023 01:24 PM - Constantin Asofiei

The setter is this:

```
   protected void setDirectField(Object value, int offset)
   {
      Runnable restoreActiveState = null;
      try
      {
         restoreActiveState = currentRecord.setActiveBuffer(RecordBuffer.this);
         OrmUtils.setField(currentRecord, offset, value);
      }
      finally
      {
         if (restoreActiveState != null)
         {
            restoreActiveState.run();
         }
      }
   }
```

#16 - 05/24/2023 01:43 PM - Constantin Asofiei

Eric, my current changes which ~~affect~~ rely on direct record access are in #7366, not in this task.

#17 - 05/26/2023 10:57 AM - Constantin Asofiei

- Status changed from WIP to Test

- % Done changed from 0 to 100

Branch 7379a was merged to trunk rev 14588 and archived. All changes intended in current task are in the branch. If something further needs to be explored, we can do in another task.

Files

buffer-binding2.png	141 KB	05/23/2023	Ovidiu Maxiniuc