Database - Feature #3958

BUFFER-COPY optimization

03/13/2019 02:01 PM - Eric Faulhaber

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

Description

Related issues:

Related to Database - Feature #4055: optimize temp-table output parameter cop...	**WIP**
Related to Base Language - Feature #4378: properly handle clob/lonchar assign...	**Closed**

History

#1 - 03/13/2019 02:15 PM - Eric Faulhaber

BUFFER-COPY

The current implementation uses reflection to invoke the getters and setters of each property of the source and destination DMO proxies. This can be expensive, due to the complex logic in the proxy invocation handler and to some degree, reflection overhead. The latter may be negligible.

This approach may not be necessary for the simple case where:

- source and destination DMO represent the same type/class;
- there are no list of fields to include/exclude from the copy;
- no-lobs is false;
- there are no assign triggers;
- any other conditions?

In this case, which may be quite common, we can use the DMO's own assign method to perform the copy, which avoids the overhead of the proxy invocation handler and of reflection itself.

I think this is even ok to do in the case where the ASSIGN clause is used, as long as no fields are excluded, or the excluded fields exactly match the fields in the ASSIGN clause. In this case, we still would do the full copy via the DMO assign method, then perform the "custom" assignment(s) afterward.

Considerations that may scuttle this idea:

- record change notification
- validation/pre-validation

BUFFER-COPY is implemented in the package private, static RecordBuffer.copy methods. There is one used for the converted language statement variant, and another for the handle method variant.

#2 - 03/26/2019 04:33 PM - Greg Shah

- Subject changed from database/persistence performance improvement ideas to BUFFER-COPY optimization

#3 - 07/08/2020 02:24 PM - Eric Faulhaber

- Related to Feature #4055: optimize temp-table output parameter copying added

#4 - 07/15/2020 12:46 PM - Greg Shah

- Related to Feature #4378: properly handle clob/lonchar assignment, especially the implicit codepage conversion added

#5 - 11/25/2020 07:14 AM - Adrian Lungu

- Status changed from New to WIP

- Assignee set to Adrian Lungu

#6 - 12/09/2020 04:15 AM - Adrian Lungu

Added a new signature for each DmoMeta - see #4055-80, #4055-81. The first one I will refer to as "quick signature", while the second is an "explicit signature". The explicit signature is suitable for operations in which a per-name property mapping is required. An example of such is below:

```
C(apple)mI(banana)I3(grapes)Z(peach)mC(pear)|2,3|2,4|4
```

The properties are sorted by their name and the unique constraints (the trailing numbers) are also sorted. Beside this, a mapping between the property id and position was built; this can check if a straight up positional copy can be done (to fasten the buffer-copy even more).

For a start I will allow fast-buffer-copy only when:

- the conditions in #3958-1 are respected
- buffers don't have before buffers
- the signatures match (same field types, names, unique constraints, mandatory constraints)

Some of the conditions above can be removed in exchange to extra-processing, other can be relaxed (as in the "quick signature").

Now I am trying to search/implement a way to assign the values from one record to another. I think that RecordBuffer.copyDMO is risky as it doesn't have any mean of notification for record changes (and record lock, dirty prop setting, validation etc.). All of these can be handled through OrmUtils.setField(BaseRecord, int, Object) which will fire BaseRecord.setDatum(int, Object). At this point I am planning to go for a OrmUtils.setAllFields(BaseRecord, BaseRecord), which can eventually allow bulk property setting, which should be smarter than the naive way (to invoke setDatum for each property sequentially).

Eric, you mentioned DMO's own assign method. I don't understand what do you mean by this; is there already a fast implementation of the above mentioned OrmUtils.setAllFields(BaseRecord, BaseRecord) or such?

#7 - 12/10/2020 11:13 AM - Adrian Lungu

I committed 3821c/rev. 11862 changes to DMOSignatureHelper which can now compute explicit signatures. The modifications from RecordBuffer allow now "fast-buffer-copy" when the conditions from #3958-6 are met. These make use of BaseRecord.setAllDatum() which is also an implementation of the idea in #3958-6. The current approach is "safe" as it can detect signature matching malfunctions; however it is naive as it calls setDatum for each property - this can be improved.

The current time performances are computed on an example with one million buffer-copy operations on temp-tables with 10 columns (warm-cold):
4GL: 1.719s - 1.786s
3821c/11862: 0.855s - 1.125s
3821c/11861: 1.721s - 2.040s

#8 - 01/04/2021 08:06 AM - Adrian Lungu

I noticed something off in the current BaseRecord.setAllDatum() implementation. This is used, according to #3958-7, in order to set multiple properties of a record at the same time. In the implementation of BaseRecord.setDatum(), I see activeBuffer and activeOffset being used - which I can't understand very well. I identified only some usages of those when using setters in RecordBuffer$Handler.invoke().

My question is: should activeBuffer be set through BaseRecord.setActiveBuffer() before attempting a buffer copy, and eventually BaseRecord.resetActiveState() afterwards? Note that now RecordBuffer.copy(DataModelObject, DataModelObject, Map<>) (which is considered to be the slow way) and RecordBuffer.fastCopy() (which is considered to be the fast way) use the "direct access" getters and setters.

#9 - 01/27/2021 11:30 AM - Eric Faulhaber

Adrian Lungu wrote:

> I noticed something off in the current BaseRecord.setAllDatum() implementation. This is used, according to #3958-7, in order to set multiple properties of a record at the same time. In the implementation of BaseRecord.setDatum(), I see activeBuffer and activeOffset being used - which I can't understand very well. I identified only some usages of those when using setters in RecordBuffer$Handler.invoke().

Sorry, I missed this question when it was posted.

BaseRecord.activeBuffer was added when I assumed there was only one path down from RecordBuffer into BaseRecord to change the value of a property (i.e., from RecordBuffer$Handler.invoke). It was only meant to be non-null for this call path, being set early in Handler.invoke, and unset at the end of Handler.invoke. This call path would only be invoked from converted business logic, either directly through a proxied setter method, or indirectly, through the runtime, such as through RecordBuffer.copy (which would also go through the DMO proxy to ensure invocation handler logic would not be skipped).

The purpose of activeBuffer originally was to ensure that we could upgrade to an exclusive record lock if necessary, when updating a DMO property's value. However, later it seems we added other call paths into BaseRecord to change property values, and this caused an NPE when activeBuffer was found to be null. Thus, code was added to bypass the lock upgrade and UNDO tracking for these call paths. I'm not sure this is correct, and at the moment, I do not recall what these other call paths are. I know Ovidiu was hitting NPEs at some point, before the bypass was added. There is still a TODO he added in BR.lockForUpdate about this.

> My question is: should activeBuffer be set through BaseRecord.setActiveBuffer() before attempting a buffer copy, and eventually BaseRecord.resetActiveState() afterwards? Note that now RecordBuffer.copy(DataModelObject, DataModelObject, Map<>) (which is considered to be the slow way) and RecordBuffer.fastCopy() (which is considered to be the fast way) use the "direct access" getters and setters.

I think RecordBuffer.copy should use BR.setActiveBuffer and BR.resetActiveState in the cases where it is not going through the RecordBuffer invocation handler, to ensure we hold an exclusive lock on the destination record (or report an error that we could not upgrade the lock), before copying into it.

#10 - 01/29/2021 07:02 AM - Adrian Lungu

- Status changed from WIP to Review

- % Done changed from 0 to 100

Added the setActiveBuffer and resetActiveBuffer with 3821c/rev. 11966. Beside those changes, I added BaseRecord.bulkDataChanged() which reports the changed data only once for all fields at once. Also, the exclusive lock will be tried only once - not for each field independently. This means that activePropPreiousValue and activeOffset does not make sense in this context. However these are used to undo the changes in case validation problems occur - these shouldn't happen in the first place due to DMO signature match guarantee. Please review.

#11 - 03/10/2021 11:00 AM - Greg Shah

What is left to do on this task?

#12 - 03/10/2021 01:22 PM - Adrian Lungu

This can be closed; the BUFFER-COPY implementation reached all the optimizations proposed here.

#13 - 03/10/2021 01:42 PM - Greg Shah

- Status changed from Review to Closed