Database - Feature #6825

improve table meta runtime (TableMapper and other)

10/06/2022 03:35 AM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Alexandru Donica	**% Done:**	80%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #6823: improve TemporaryBuffer/RecordBuffer def...	**New**

History

#1 - 10/06/2022 03:36 AM - Constantin Asofiei

- Related to Feature #6823: improve TemporaryBuffer/RecordBuffer define and buffer delete added

#2 - 10/06/2022 03:37 AM - Constantin Asofiei

In TableMapper, because of some mutable state, a new LegacyTableInfo was being created for each buffer definition. This will be refactored in 6129a/3821c, to keep a cache of LegacyTableInfo, and all mutable field state kept in MutableFieldInfo instances.

This needs further analysis (in conjunction with #6823) to improve the static temp-table definition and the buffer definition in general.

#3 - 10/06/2022 04:27 AM - Constantin Asofiei

Some changes are in 3821c/14272. Ovidiu, please review.

#4 - 01/05/2023 02:02 PM - Constantin Asofiei

The MutableFieldInfo is being kept in TableMapper$TempTableMapper.fieldInfo by the TempTable instance. So I see no reason not to move this Map<String, MutableFieldInfo> mapping of legacy field names to their mutable info directly as a field in AbstractTempTable. This solves some unnecessary map and context local lookup, and lazily populates this map each time the runtime tries to get (or set) the info for a certain field.

But, there is an issue: DataSet.copyTempTableOptions will copy the mutable field info from the source to the destination, even if the source has not 'mutated' this info at all! So the destination will think that all information has been 'mutated'.

I'm thinking of an approach where:

- unless a setter for this mutable info is being called by the application, do not create any MutableFieldInfo mapping and just use the 'template' from the TempTable's LegacyTableInfo.p2j field info
- any runtime getter/setter must check if this info is being actually changed (mutated) - unless this happens, do not create MutableFieldInfo instances, and rely on the template at LegacyTableInfo.p2j field info.
- obviously, any getter call will never create MutableFieldInfo instances, and will just default to LegacyTableInfo.p2j if one does not exist.

Ovidiu, any concerns on this?

#5 - 01/05/2023 02:20 PM - Ovidiu Maxiniuc

Totally agree!

#6 - 01/10/2023 07:01 AM - Constantin Asofiei

The changes for #6825-4 are in 6129b/14347

#7 - 01/20/2023 09:53 AM - Constantin Asofiei

- % Done changed from 0 to 80

The approach in #6825-4 improved the TableMapper memory and CPU footprint, but the access to these is still done in a 'heavy way' - this is especially when BUFFER-FIELD resources are required for all columns in a temp-table with 100s of columns. Some more analysis is required.

#8 - 01/05/2024 03:48 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Alexandru Donica

Alexandru [ad], please attempt to continue profiling and optimization of TableMapper. The recent changes shifted a lot of work from heavy TableMapper to DmoMeta (where possible). Lets see if we can continue this approach.

Please profile the large POC we have to check how TableMapper still behaves. Also, lets target BUFFER-FIELD construct even deeper. I am also aware of cases where BUFFER-FIELD is used inside a loop across all fields of a table (either from the converted code or from our run-time). Lets see if we can optimize this. **Don't disregard conversion changes** - we collapsed FOR EACH tt: DELETE TT. constructs into EMPTY-TEMP-TABLE. Maybe we can do the same for heavy time consuming flows.

#9 - 01/30/2024 12:50 PM - Alexandru Lungu

Alexandru [ad], AFAIK, you already have some changes on TableMapper to improve BufferFieldImpl.asP2JField. This is a very used function when using DataSet.createLikeImpl.

Please refer to #7026-310. Most of the data-set related methods are quite slow in FWD alone. Some of them are related to TableMapper (like in the createLikeImpl case). Please extend your research and scope of this task #6825 to better fit the bottlenecks found in #7026 (especially in regard to datasets).