Database - Bug #5176

Buffer field attribute leak

03/03/2021 07:07 PM - Ovidiu Maxiniuc

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

Description

Related issues:

Related to Database - Feature #3574: finish implementation of temp-table XML ...	**Closed**

History

#1 - 03/03/2021 07:09 PM - Ovidiu Maxiniuc

- Related to Feature #3574: finish implementation of temp-table XML support added

#2 - 03/03/2021 08:10 PM - Ovidiu Maxiniuc

I will try to summarize an ugly issue I am investigating lately.

I was working on a issue related to #3574-11 and I discovered that the attribute values for buffer fields are not limited to dynamic temp-tables only. At first I assumed that, since we use the DmoMeta object as key for storing structures in TableMapper and instances of the dynamic temp tables share it, only they are affected. My first attempts was to change the static accesses to TableMapper internal data. The permanent tables did not seem to suffer from this issue, and the static temp-tables. The only solution I imagined was to pass (or extract) somehow the TempTable object for dynamic temp-tables accessed. The problem is TableMapper is constructed quite well structured and the interface is almost consistent of the table being queried. Since the permanent tables do not have a parent temp-table, all invocations to TableMapper methods should have been split in two: one method with current signature (DmoMeta or DMO Class<> interface) for permanent tables and one method to pass along the parent temp-table reference to be used as lookup for correct LegacyTableInfo / LegacyFieldInfo. Since there are a lot of these, I abandoned the idea. It is not a good OO code to use conditional to detect the method to be called.

At this moment I tried to approach the problem from another angle. What if I remove the cause for this issue - the shared DMO/Interface? If we are using different LegacyTableInfo for tables sharing the same DmoMeta, what would be the penalty for dropping the interface-keyed cache? Some extra CPU cycles for constructing the DMO interface each time. And some memory to store this duplicate objects. It apparently worked at first but then I had the idea of 'hitting' the static temp-tables with similar constructs. There was nothing to suggest any flaw in this regard.

So I constructed a procedure which modifies the field attributes of a static temp-table parameter in called procedure. It looks like this:

```
/* master.p */
DEFINE VARIABLE hSlave AS HANDLE NO-UNDO.
DEFINE TEMP-TABLE tt FIELD dex AS DECIMAL.

MESSAGE "Outer Before:"
        BUFFER tt:BUFFER-FIELD("dex"):DECIMALS
        BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-NAME
        BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN.

RUN slave.p PERSISTENT SET hSlave.
RUN addRecord in hSlave(OUTPUT TABLE tt APPEND).

MESSAGE "Outer After:"
        BUFFER tt:BUFFER-FIELD("dex"):DECIMALS
        BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-NAME
        BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN.
```

```
/* slave.p */
```

```
DEFINE TEMP-TABLE tt FIELD dex AS DECIMAL.
DEFINE TEMP-TABLE tt1 LIKE tt.

PROCEDURE addRecord.
   DEFINE OUTPUT PARAMETER TABLE FOR tt1.

   CREATE tt.
   ASSIGN tt.dex = 3.1415.

   CREATE tt1.
   BUFFER-COPY tt to tt1.

   MESSAGE "  Inner Before:"
           BUFFER tt:BUFFER-FIELD("dex"):DECIMALS
           BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-NAME
           BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN
           BUFFER tt1:BUFFER-FIELD("dex"):DECIMALS
           BUFFER tt1:BUFFER-FIELD("dex"):SERIALIZE-NAME
           BUFFER tt1:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN.

   BUFFER tt:BUFFER-FIELD("dex"):DECIMALS = 5.
   BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-NAME = "meeeh".
   BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN = true.

   MESSAGE "  Inner After:"
           BUFFER tt:BUFFER-FIELD("dex"):DECIMALS
           BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-NAME
           BUFFER tt:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN
           BUFFER tt1:BUFFER-FIELD("dex"):DECIMALS
           BUFFER tt1:BUFFER-FIELD("dex"):SERIALIZE-NAME
           BUFFER tt1:BUFFER-FIELD("dex"):SERIALIZE-HIDDEN.
END.
```

Running master.p procedure on OE will print:

```
Outer Before: 0 dex no
  Inner Before: 0 dex no 0 dex no
  Inner After: 5 meeeh yes 0 dex no
Outer After: 0 dex no
```

It's evident that the attribute changes affected only the local tt table in slave.p procedure. tt1 parameter table is not affected, and neither the tt table in master.p.

When running the same piece of code in FWD we get:

```
Outer Before: 0 dex no
  Inner Before: 0 dex no 0 dex no
  Inner After: 5 meeeh yes 0 dex no
Outer After: 5 meeeh yes
```

The changes in local tt table in slave.p were picked by the tt table in master.p. This is because, like the dynamically constructed temp-table, these also share the same DMO interface so they are mapped to same LegacyTableInfo / LegacyFieldInfo in TableMapper.

But wait, there is more.
While the dynamic temp-tables which share the same interface are automatically dropped as soon as the user context is dropped (the DynamicTempTableMapper dynamicDB is ContextLocal in TableMapper), the StaticTempTableMapper staticTempDB is just static. So running the same master.p again (or in parallel), we get:

```
Outer Before: 5 meeeh yes
  Inner Before: 5 meeeh yes 0 dex no
  Inner After: 5 meeeh yes 0 dex no
Outer After: 5 meeeh yes
```

All three buffer fields attributes (DECIMALS, SERIALIZE-NAME, SERIALIZE-HIDDEN) leaked from a table to another sharing the same interface, but also to another user.

To be honest when I writing the ABL code I was expecting to observe something like: Inner After: 5 meeeh yes 5 meeeh yes, I.e to see the leak behaving between the tt and tt1 from server.p, since the definition of tt1 LIKE tt would make them having the same structure and logically make them share the same DMO interface. Apparently, this is not the case. For the moment I do not have the answer why this is not happening.

#3 - 03/04/2021 08:02 AM - Greg Shah

My understanding is that in the 4GL every temp-table **instance** is a completely separate table. Multiple table instances are created possibly (but not always) from the same definition. Using table parameters is the most common reason for this. This is why they have all the BIND, APPEND, BY-REFERENCE features since by default table parameters cause new instances to be created.

Each of these instances have their own metadata, so changing one will not affect any others even though they were created from the same 4GL code definition.

This approach helps explain why they were so loose with their copying and parameter passing. Since the 4GL syntax never created any mechanism to expose a single definition of a temp-table, each compile unit must have its own internal definitions. Combined with the concept that at runtime, each temp-table is a separate instance with its own metadata, it naturally led to an approach best described as "is this table compatible enough to interoperate". As shown in #3751-492, table parameters are compatible at the structural level (field type, order, number and extent), no table/field names or other options are considered. Two completely different table definitions that have the same structure are completely compatible for passing as parameters. It is certainly a bad/confusing/error prone practice to write code that way but it is the way the 4GL works.

In FWD, we have always been trying to force a different model. It is why we are spending so much time trying to implement TT features because we are not properly matching the 4GL design. For temp-tables, they have no concept of a common DMO. Even for permanent tables, I suspect we have a problem when you consider that in the 4GL, each instance of a database is independently managed. Where we have a single DMO for all permanent database instances.

#4 - 03/09/2021 11:53 AM - Eric Faulhaber

Ovidiu, if every buffer has its own writable version of a certain set of attributes, does it make sense for each buffer to have its own DmoMeta instance, and to use the instances we create upon startup as templates for these buffer-specific instances?

#5 - 03/09/2021 02:44 PM - Ovidiu Maxiniuc

I have really thought of this issue during last days.

The DmoMeta carries immutable data, that is: information which is used to create the buffer and configure it with the start-up attribute values. If two tables share the same DmoMeta instance (as far as I know, only related static temp-tables and dynamic temp-tables can do this), they will extract the information from this "prototype" or "template" each time is needed. But the TableMapper.LegacyTableInfo, TableMapper.LegacyFieldInfo, etc will store the alterable values for each table instance (regardless of its type). If the attributes are mutable, then FWD will allow to change them, but only for the lifetime interval of the specific table. This hold only for temp-tables, as the attributes of permanent tables are not mutable (AFAIK).

So, not each buffer has a writeable version of the DMO meta data. The permanent database buffers can share the same DmoMeta, even if they are from different user contexts, but the dynamic temp-table buffer must be really isolated from this POV. The above mentioned TableMapper structures do exactly that.

Status report:
In my isolated testcase posted in this thread, the issue seems fixed now, I am extending my tests to xfer test suites and later to customer application.

#6 - 03/10/2021 04:27 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

I committed the fix in 3821c as revision 12108. The update is a aggregate, it contains fixes for other smaller issues. I fixed a few issues which I identified while testing in customer application.

Eric, please do a quick review.

#7 - 03/12/2021 11:39 AM - Eric Faulhaber

Code review 3821c/12108:

Generally, the changes look good to me, though the change was so dense in TableMapper, it is difficult to know for sure with just a review. There were also a number of changes outside the persist package that seemed unrelated to this issue. There are TODOs in BufferFieldImpl.getDefaultString about not caching transient results. What is doing the caching and what is the plan for these?

Is there anything left to do for this issue?

#9 - 03/15/2021 01:35 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Code review 3821c/12108:
> Generally, the changes look good to me, though the change was so dense in TableMapper, it is difficult to know for sure with just a review.

The main idea is that all APIs which had a DMO class or a Buffer as parameter were changed to accept the TempTable or RecordBuffer instead. This way the TableMapper is able to detect the exact LegacyTableInfo to use/return.
There are some other pieces of code which I think can be replaced by direct delegation to TableMapper: RecordBuffer.setFieldLabel(), RecordBuffer.setFieldValidateExpression(), and related, and the field* from that RecordBuffer removed. However, I did not do this because I was afraid the changes would be too radical. There is this comment in RecordBuffer:7255 about a *Progress quirk*. I do not know how to test the new code to see whether it would work as expected. There might be some test cases. Nonetheless, I think this support code was written for exactly the same reason I did the changes in TableMapper.

> There were also a number of changes outside the persist package that seemed unrelated to this issue.

Yes, the commit contains other fixes, as noted in the comment, most of them related to #5034.

> There are TODOs in BufferFieldImpl.getDefaultString about not caching transient results. What is doing the caching and what is the plan for these?

Actually I was mislead here. The documentation upholds that the DEFAULT-STRING attribute has Access: Readable/Writeable. It is not true; from my tests, not even for dynamic temp-tables. But the implementation of getDefaultString is incorrect. It does not always return the "unformatted version of the INITIAL attribute".

> Is there anything left to do for this issue?

No, I think this can be closed. I will eventually create a new thread to track the above issues.

#10 - 03/17/2021 02:29 PM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed