Database - Bug #4825

OutputTableHandleCopier dynamically defines destination temp-table

07/29/2020 01:55 PM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	
Description			

History

#1 - 07/29/2020 02:24 PM - Eric Faulhaber

As I was debugging an API-based regression test, I noticed we end up quite often in this code in OutputTableHandleCopier.finished:

```
      if (calling.getTableHandle()._isValid())
      {
         ... // copy all rows between existing temp-tables
      }
      else if (called._isValid() && called.unwrapTempTable().prepared().booleanValue())
      {
         if (!calling.isRemoteParameter())
         {
            // create new table
            TempTableBuilder builder = new TempTableBuilder();
            res = builder.copyTempTable(called,
                                        calling.isAppend(),
                                        false,
                                        false,
                                        "").booleanValue();
            ...
```

TempTableBuilder.copyTempTable copies not only the records, but the entire structure of the temp-table. From AbstractTempTable.copyTempTable:

```
         ...
         boolean res = createLike(other).booleanValue();
         if (!res)
         {
            return new logical(false);
         }
         StringBuilder tableName = new StringBuilder(prefix.isUnknown() ?
                                                     DEFAULT_COPY_TEMP_TABLE_PREFIX :
                                                     prefix.getValue());
         tableName.append(otherBuf.getParentTable().name().getValue());
         tempTablePrepare(tableName.toString());
         ...
```

I have not instrumented the code to measure the time this takes, but it seems extremely expensive to do it this way, when all we really want to do is have a new virtual temp-table with exactly the same structure as the source temp-table.

If I understand the logic correctly, we are not in a remote call situation, so we already should have the DMO interface and class created and loaded for the destination table type we need. It seems all we need is to multiplex the existing temp-table to create a new virtual temp-table into which to copy the records. This essentially means we only should need to define a new buffer for the source table's DMO type and open its scope to get a new multiplex ID. I can't think of any reason we should be recreating the wheel for the destination temp-table every time we go through this logic.

#2 - 07/29/2020 02:36 PM - Constantin Asofiei

Eric, I understand your POV, but at the caller's side we have a handle parameter, right? In this case, we need to create a copy of the TEMP-TABLE 4GL resource, too. Not just the buffer. As that's what the handle needs to have - a reference to a new 4GL-compatible temp-table resource.

This already will create the dynamic DMO only once. What could be optimized is tempTablePrepareImpl in case when is for a copyTempTable call - to bypass all the validation, as we already know the structure is valid.

#3 - 07/29/2020 04:19 PM - Constantin Asofiei

Something else we could do is to force the DMO interface to be the one from the source temp-table (to avoid building a dynamic DMO). And do a fast 'prepare' for the copy-temp-table case. The different multiplex ID should be computed automatically, as we already do when a dynamic DMO is shared between dynamic temp-tables.

#4 - 07/29/2020 05:19 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Something else we could do is to force the DMO interface to be the one from the source temp-table (to avoid building a dynamic DMO). And do a fast 'prepare' for the copy-temp-table case. The different multiplex ID should be computed automatically, as we already do when a dynamic DMO is shared between dynamic temp-tables.

I initially did not realize the need for the legacy resources, so I understand my original proposal may have been too aggressive. But yes, I think we still could short-circuit a lot of the work that is happening in the copier. It seems to me that the createLike implementation in general could be a lot more efficient, in the case where we are not overriding anything about the source table, and we already have the DMO interface and class generated, either as a static table or as a dynamic temp-table which previously was prepared.

Other than the integration with TRPL at runtime, I haven't looked deeply into the dynamic temp-table building process, but it seems we should be able to cache the legacy resources that come out of this process by DMO type or some other key, when we are just copying the structure of a temp-table that already was built.

A useful first step to help prioritize this optimization would be to instrument this code with high-resolution timing, so we can determine how much time we really are spending doing this work. I expect the first time through for a particular table structure is quite expensive. I also suspect that the subsequent times through are faster (since the conversion results are cached), but still a lot slower than we might like.

#5 - 10/19/2020 04:29 PM - Eric Faulhaber

- Assignee set to Constantin Asofiei

Eric Faulhaber wrote:

> A useful first step to help prioritize this optimization would be to instrument this code with high-resolution timing, so we can determine how much time we really are spending doing this work. I expect the first time through for a particular table structure is quite expensive. I also suspect that the subsequent times through are faster (since the conversion results are cached), but still a lot slower than we might like.

Constantin, can you take the next step looking into this?

#6 - 10/20/2020 01:21 PM - Constantin Asofiei

- File table_handle_copier_jmx_timer.txt added

Attached is the patch which instruments the OutputTableHandleCopier.finished method via JMX bean.

Files

table_handle_copier_jmx_timer.txt	1.66 KB	10/20/2020	Constantin Asofiei