Database - Support #6713

check if it is faster to use separate temp-table instances instead of multiplex id

08/25/2022 12:17 PM - Greg Shah

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Dănuț Filimon	**% Done:**	70%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	
Description			

History

#1 - 08/25/2022 12:18 PM - Greg Shah

It seems like for tables with very large numbers of records might be slow when using a multiplex approach as opposed to separate table instances for each multiplex id. We can check this to see the difference.

#2 - 09/02/2022 09:06 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Dănuț Filimon

#3 - 09/07/2022 01:52 PM - Ovidiu Maxiniuc

The investigation should take into account that, beside the expected direct SQL internal benefits, should also include the following aspects:

- for each multiplexed level a new temp-table should be dynamically created and dropped when the multiplex's scope ends. We are already forced to use a naming schema for temp-tables as <converted-original-name>_<N>_<M>. To avoid naming conflicts, the _multiplex will also need to be added as a decoration in the SQL table name, so the new naming schema would be <converted-original-name>_<N>_<M>_<_multiplex>. Additionally, the secondary tables (for extent fields) must also created/dropped.
- when the multiplexed scope ends the table is simply dropped, instead of deleting all its records;
- the construction of SQL statements for insert/select/update/delete are a bit simpler, as the term and _multiplex=? is no longer present.

The good news is that this investigation only needs to be performed on H2 (the dedicated dialect for temp-tables).

#4 - 09/07/2022 03:52 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> We are already forced to use a naming schema for temp-tables as <converted-original-name>_<N>_<M>. To avoid naming conflicts, the _multiplex will also need to be added as a decoration in the SQL table name, so the new naming schema would be <converted-original-name>_<N>_<M>_<_multiplex>.

I thought this naming convention was just for the DMO interfaces. When did we start using this naming convention for the database temporary tables themselves? AFAIK, the temporary tables in H2 are just named tt1, tt2, ... tt<N>. Am I just thinking of dynamic temp-tables? I thought those were named dtt1, dtt2, ... dtt<N>

Frankly, I don't see how using separate tables could be faster, what with all the extra table creates and drops that would be needed. Also, IIRC, "deleting" records in a subtransaction was mapped to setting the (positive) multiplex ids of deleted records to their additive inverses. Any such code would have to be reworked to perform actual deletes.

If using a multiplex ID is slower than separate tables (at least for selects), it suggests to me a problem with the way the multiplex IDs are defined in indices and used in queries. If that is the case, we should address that.

I'm not saying we should not research and compare relative costs of the different approaches, but we probably need to do this in an abstract way first, before making any changes to the FWD runtime, because such changes would be pretty invasive.

#5 - 09/07/2022 04:16 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> I thought this naming convention was just for the DMO interfaces. When did we start using this naming convention for the database temporary tables themselves? AFAIK, the temporary tables in H2 are just named tt1, tt2, ... tt<N>. Am I just thinking of dynamic temp-tables? I thought those were named dtt1, dtt2, ... dtt<N>

You are right, thank you for correcting my mistake. Indeed, the SQL table names for static temp-tables use tt<N> sequence and those dynamically constructed - dtt<N>. Yet, if separate tables are to be created, they still need to carry the multiplex id in their name.

> I'm not saying we should not research and compare relative costs of the different approaches, but we probably need to do this in an abstract way first, before making any changes to the FWD runtime, because such changes would be pretty invasive.

I agree with that.

#6 - 09/14/2022 11:29 AM - Alexandru Lungu

Please consider several "abstract" testcases (eventually with etime) which can show clear time differences in different scenarios. Run once for multiplex (tables with same structure and name) and once without (tables with different name). For the latest, only try to suppress _multiplex field in the generated SQLs (no need for further changes for now).

- many simple operations on few static tables: 5 tables (5 fields) with thousands of inserts/reads/updates/deletes (adjust numbers for retrieving proper time measurements)
- several simple operations on many static tables: 20 tables (5 fields) with hundreds of inserts/reads/updates/deletes (adjust numbers for retrieving proper time measurements)
- experiment with bigger tables (>20 fields)
- some tests with dynamic tables; need to investigate how multiplex is used in such cases (only if the tables have the same structure?)
- complex queries on tables with several multiplex collections
    - **run several persistent procedures to populate a single converted temporary table with several multiplex values**
    - run different queries (using WHERE, client WHERE, ORDER BY, JOIN, etc.) using this table
    - try PRESELECT and FOR

    - try different indexes
- using the same "fat" table, experiment:
    - bulk delete from table
    - COPY-TEMP-TABLE with different parameters (like append-mode)
    - use the table as procedure parameter

For non-multiplex case, you can give the temporary tables different names such that they are not compressed in the same physical table in FWD.

#7 - 09/16/2022 09:48 AM - Dănuț Filimon

- File table_multiplex_no-multiplex.png added

- File table_average_simple_operation.png added

Alexandru Lungu wrote:

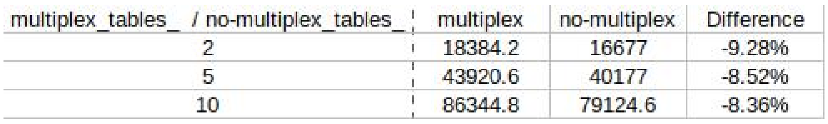
> Please consider several "abstract" testcases (eventually with etime) which can show clear time differences in different scenarios. Run once for multiplex (tables with same structure and name) and once without (tables with different name). For the latest, only try to suppress _multiplex field in the generated SQLs (no need for further changes for now).

I made 4GL programs for each case (tables with the same structure and name and tables with different name). I also created a copy of 3821c and started making a running version that does not use multiplex and succeed (it works for the current examples that I use, but might need further changes when converting other complex 4GL programs).

Using the ETIME method, I tested both multiplex and non-multiplex converted 4GL programs that have the following specifications:

- **multiplex_tables_{n}**: 1 main temporary table, **n** temporary tables and **n** main table copies (same name and structure) in n different files.
- **no_multiplex_tables_{n}**: 1 main temporary table, **n** temporary tables and **n** main table copies (different name, but same structure) in n different files.
- each temporary table has an initial insert of 1000 entries
- persistent procedures were used
- the tests performed an INNER JOIN between the temp-table and main table copy in each file.

I used the average of 5 sets of results and obtained the following:



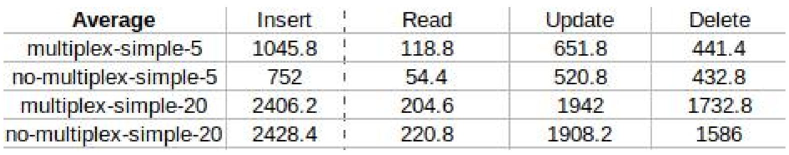
multiplex_tables_ / no-multiplex_tables_	multiplex	no-multiplex	Difference
2	18384.2	16677	-9.28%
5	43920.6	40177	-8.52%
10	86344.8	79124.6	-8.36%

The results are measured in milliseconds.

Each temporary table has an initial insert of 1000 entries with random data, this means that the statements executed between a multiplex and no-multiplex test are different. That being said, I reduced the scale of the initial insert and used fixed data in order to track any difference between the results of both tests. I confirmed that the same logic is used between the SQL statements that use multiplex and the SQL statements of the non multiplex test.

> many simple operations on few static tables: 5 tables (5 fields) with thousands of inserts/reads/updates/deletes (adjust numbers for retrieving proper time measurements)
> several simple operations on many static tables: 20 tables (5 fields) with hundreds of inserts/reads/updates/deletes (adjust numbers for retrieving proper time measurements)

I created examples for both cases. Each example inserts 1000 entries, reads them, updates the fields of each one and then deletes them. These tests were done similarly to the previous ones, using the ETIME method and calculating the average of 5 sets of results.



Average	Insert	Read	Update	Delete
multiplex-simple-5	1045.8	118.8	651.8	441.4
no-multiplex-simple-5	752	54.4	520.8	432.8
multiplex-simple-20	2406.2	204.6	1942	1732.8
no-multiplex-simple-20	2428.4	220.8	1908.2	1586

In the test that used 5 tables, there is a difference between operations, but the test with 10 tables shows mostly equal times between them. I think this test needs to be done again, this time using at least 3000-5000 entries as an initial insert.

#8 - 09/23/2022 09:48 AM - Dănuț Filimon

I made a set of tests for multiplex and no-multiplex using indexes.

The tests have the following specifications:

- **multiplex-no-index-{n}, multiplex-one-index-{n}, multiplex-two-index-{n}**.
- **no-multiplex-no-index-{n}, no-multiplex-one-index-{n}, no-multiplex-two-index-{n}**.
- Each test using multiplex has **1** main temporary table and **n** main table copies (same name and structure) in **n** different files.
- Temporary tables have 5 fields (1 integer and 4 character).
- The tests that don't use multiplex use main table copies with different names.
- Each test consists of executing an initial **insert** of 5000 entries in each temporary table, **read** (using index if it exists), **update**, **update with one sort and with two sorts** (by fields that are not indexed).
- Results were measured using ETIME.

Here are the results for using 5 and 20 files:

	multiplex	no-multiplex	Difference (%)
n = 5, no-index	9000.66	9157	+1.74%
n = 20, no-index	30862	29461.33	-3.96%
n = 5, one-index	11783.33	12192	+3.47%
n = 20, one-index	41981	40099.33	-4.48%
n = 5, two-index	10742	11128.66	+3.60%
n = 20, two-index	37829.33	36673	-3.06%

When comparing the results between having a small number and a large number of files, the results are better for 20 files. There is not much of an improvement when using indexes.

I wrote other tests similar to the previous ones, but using dynamic tables and no indexes. The results were:

	multiplex	no-multiplex	Difference (%)
5 files	40422.33	40302.33	-0.30%
20 files	138964.33	139375.66	+0.30%

There is also no measurable difference between multiplex and no-multiplex when using dynamic tables. I also need to mention that a normal update takes 10x more time than an update using sort and it represents 60-70% of the time for each test, this explains the time obtained.

I rewritten the tests in #6713-7 because I improved/corrected parts of the code.
The specifications are:

- Tests were done for temporary tables with 5 fields and then 20 fields.
- Each test consists of executing an initial **insert** of 5000 entries in each temporary table, followed by **read**, **update** and **delete**.

5 fields	Create	Read	Update	Delete
multiplex, 5 files	1254.33	305.66	1971.66	42.33
no-multiplex, 5 files	1475.66	369.33	1963.66	35.33
multiplex, 20 files	3563	514.66	6858	109.66
no-multiplex, 20 files	3365	677.33	6274	85.33

20 fields	Create	Read	Update	Delete
multiplex, 5 files	1561.33	373.66	8096	41
no-multiplex, 5 files	1804.33	553	7919.66	32.66
multiplex, 20 files	4115.33	666.66	27736.66	102.66
no-multiplex, 20 files	3910.33	868.66	26731.33	84.33

When faced with multiple files, using temp-tables instances instead of multiplex is a bit better when creating, updating and deleting. At the same time, reading data has a decreased performance in all cases. I don't think that using temp-tables instances is better at the moment by looking at the current tests since reading data is more frequent than any other operation and most improvements are don't have a considerable margin.

Alexandru Lungu wrote:

- using the same "fat" table, experiment:
    - bulk delete from table
    - COPY-TEMP-TABLE with different parameters (like append-mode)
    - use the table as procedure parameter

I will continue to test the following and possibly use even larger tables for previous tests if any improvement is shown.

#9 - 10/07/2022 09:11 AM - Dănuț Filimon

- % Done changed from 0 to 70

I made tests for bulk delete and copy temp-table.

- **copy-temp-table-multiplex-{n}**, **copy-temp-table-no-multiplex-{n}**.
- **delete-temp-table-multiplex-{n}**, **delete-temp-table-no-multiplex-{n}**.
- Each test has **1** main temporary table and **n** main table copies in **n** different files.
- Temporary tables have 50 fields (1 integer and 49 character).
- The tests for copy-temp-table consist in an **initial insert** of 1000 or 5000 entries in each temporary table and a **normal-copy** made between the main table and each temporary table. A handle is created for the main table, which is the target handle that is sent to each file as an INPUT-OUTPUT parameter. In each file there is a source handle created in the same manner and is used by the procedure that receives the target handle to copy the records of the temporary table from the file to the main table.
- The copy is done using **COPY-TEMP-TABLE**.
- The tests for delete-temp-table consist in an **initial insert** of 5000 entries and a **bulk delete** done using **EMPTY TEMP-TABLE**.

The results for the **bulk delete** test were:

Bulk Delete Temp-table			
Test	multiplex	no multiplex	Difference
5 tables	51.667	33	-36.129%
20 tables	116.333	78.66	-32.383%

In this case, not using multiplex is better, with a difference greater than 30%.

However, the **copy temp-table** results were not significant.

Copy Temp-table				
Normal Copy	multiplex		no multiplex	
Test	1000	5000	1000	5000
5 tables	53	167.333	67.333	169.333
20 tables	149.667	451.667	131.333	449.333

The results are not good for a small number of entries and for a larger number, we can see that the margins are not big.

I've also tried to test the APPEND-MODE as suggested, but I encountered problems when testing the no-multiplex case that would require modifications in how the append-mode works. In this case, when using append-mode, a dynamic table is used throughout all operations. When an append is finished, the records belonging to the table that was appended are **not deleted** from the dynamic table, this action is done after all appends are done and it uses the multiplex id to delete those specific records. This means that each time an append is done, the number of records in the dynamic tables **doubles**.

Most of the tests showed almost equal results between multiplex and no-multiplex, expect bulk delete. I don't think that using temp-table instances instead of multiplex id would make a significant change.

Committed testcases/rev.2367. I added a smaller version of each test (simple operations, indexes, dynamic table, bulk delete and copy temp-table) in testcases.

Files

table_multiplex_no-multiplex.png	9.82 KB	09/16/2022	Dănuț Filimon
table_average_simple_operation.png	10.6 KB	09/16/2022	Dănuț Filimon