Database - Feature #7404

Trasform replace-mode into append-mode when target table is empty

06/01/2023 08:11 AM - Alexandru Lungu

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

Description

Related issues:

Related to Database - Bug #7403: Copy-temp-table with replace-mode should rep...	**Review**
Related to Database - Bug #7488: Slow fast-copy with before tables in H2	**Test**
Related to Database - Bug #7489: Lazily initialize DMO signature	**WIP**

History

#1 - 06/01/2023 08:30 AM - Alexandru Lungu

In FWD, copy-temp-table append mode is faster as it can be resolved using fast-copy (INSERT INTO SELECT FROM).

There are several customer applications that use replace mode (some that use it more often than append mode). I've also seen that a consistent number of such replace copies are over an empty target table. This means that the replace-mode should transform into an append-mode (as there is nothing to replace).

However, this is a bit tricky as 4GL actually replaces the already copied records! Fast copy however works only with matching signature, so it is guaranteed that the source table has the same unique constraints as the target table. Therefore, we can still do the conversion from replace-mode to append-mode. This constraint is something to think about when we will allows copies with non-matching signature.

The task here should be of testing this approach with large customer applications, profilers, etc.

#2 - 06/08/2023 08:01 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Ștefan Roman

#3 - 06/08/2023 11:12 AM - Alexandru Lungu

Ștefan, I will put you up-to-date with the FWD implementation of copy-temp-table:

4GL logic
copy-temp-table is used to copy data (all rows) from one temporary table to another. It can use different options like below. Note that the source and destination should match the schema (not the field names). Basically, the tables should have the same number of field of the same types in the same order.

- no option: clear the destination and copy all from source
- append: don't clear the destination and append from source. If a unique index is violated, the copied row is skipped.
- replace: don't clear the destination and replace records with the same values on the primary unique index. Rows that don't replace anything are just appended
- loose-copy-mode: don't do an eager match of the schema; just copy what you can based on field names. Every field that exists in the source and

target and have the same data type will be copied, the other will be set on the default value in the destination.

FWD logic
We do this in TemporaryBuffer.copyAllRows. In fact, we use this method for many other operations like input-output table parameters. The "classic" and slow way of doing the copy is by extracting all rows from the source and inserting them one-by-one in the destination. This was the initial implementation and works for any options (append / replace / loose-copy-mode etc.). However, there was some effort to optimize using "fast-copy", which makes use of INSERT INTO [...] SELECT FROM [...] SQL syntax. Of course, this doesn't work in all cases, but most of them are handled this way. To trigger a "fast-copy" you need:

- there is no option, so this is quite a simple copy
- append is used, so this is a bit complex, but still can work for schema matching tables, including unique indexes. You can do fast-copy only if the destination is "more restrictive" than the source.
- replace mode is **not** supported (mainly because it is more complex than a simple INSERT INTO SELECT FROM + the FWD cached DMO should be invalidated)
- loose-copy-mode is supported
- denormalized and normalized extents are supported
- before tables are supported

Copy-temp-table complexity
The process of copy is not trivial as there are several aspects to take care:

- extent fields (normalized or denormalized)
- before tables
- unique constraint validation
- note that triggers are **not** available for temp-tables, so not need to worry about that
- in replace-mode, we need to update the cached DMO which were replaced
- loose-copy-mode is clearly not trivial

This is why you will find quite a consistent code in copyAllRows - make sure you get a good grip on it before attempting to change.

Multiplex
Temp-tables which share the same name and schema in 4GL are grouped inside the same physical table in FWD. That is, if you have 3 tt tables across the converted application, H2 will create only one tt table with a _multiplex column that will discriminate between the legacy tt tables. There is no logic that requires queries cross-multiplex. Therefore, all queries / dml should have something like _multiplex = ? in their WHERE clause. Also, note that the first field in all temporary table indexes start with _multiplex out of obvious reasons.

You will also notice other fields that start with _. These are reserved columns used by FWD and do not exist in 4GL. Some of these should be changed after copy-temp-table.

Extents
For a while, FWD modeled the extent fields outside the "master" table, in tables named for instance tt__5 (for extents of size 5). tt__5 has 3 columns: peer (fk to master table), index (the extent index), value (the value at the index inside the extent). This technique is named "normalization". At copy-temp-table, you need to copy the records from source__5 to destination__5 as well, based on the peer column.

At some point, FWD supports de-normalized extent fields, so that the extent fields are embedded in the master row. Basically, you will expect f1, f2, f3, f4 and f5 fields to represent the values of a f extent 5 field. For this technique, it is easier to do the copy-temp-table as you don't have a separate tt__5 table to deal with.

Before
4GL allows the definition of before tables for temp-tables. Basically, they store a state before certain operations in the "master" table. On copy-temp-table, the before table should be copied as well. This is quite easy to manage, as the copy process is usually identical to before tables (including extents). The only complication appears when you need to remap the _peerRowid between the master table and before table, so that the newly copied records reference each other (master - before).

#4 - 06/13/2023 03:29 AM - Alexandru Lungu

- Related to Bug #7403: Copy-temp-table with replace-mode should replace already inserted records added

#5 - 06/14/2023 04:43 AM - Alexandru Lungu

Created 7404a.

#6 - 06/14/2023 05:48 AM - Ștefan Roman

Committed to 7404a revision 14626.

#7 - 06/14/2023 08:02 AM - Ștefan Roman

Fixed an error and committed again on 7404a revision 14627.

#8 - 06/20/2023 02:07 AM - Alexandru Lungu

- % Done changed from 0 to 100

- Status changed from WIP to Review

Review of 7404a revision 14626

- I am OK with the functional changes.
- Please add a comment before the fast-copy if conditional: // if the destination is empty, replace works as append
- Align the return true; statement properly.

#9 - 06/20/2023 02:20 AM - Ștefan Roman

I added the changes and committed again on 7404a revision 14628.

#10 - 06/21/2023 06:10 AM - Alexandru Lungu

- % Done changed from 100 to 60

- Status changed from Review to WIP

My profiling still shows slow code here - even slower than before with +1.3%. In fact, this is because of the following two points:

- replace-mode triggers loose-copy-mode due to a 4GL quirk. However, most of my examples are actually simpleCopy = srcBuf.getDMOImplementationClass().equals(dstBuf.getDMOImplementationClass()), so the schemas match exactly. From my POV, this implies that we can **always** set loose-copy-mode on false if simpleCopy is true - the order of the fields, types and indexes will always match exactly. In FastCopyHelper, if simpleCopy is true, canFastCopy fails because it doesn't have propsMap to do the proper uniqueness check. Therefore, most of the times, the change from #7404-8 is not hit.
- FastCopyHelper seems to be quite slow with loose-copy-mode. We shall set this on false whenever possible, even when the DMO interfaces don't match. What if the explicit signatures match? I guess this means we can do the fast-copy without loose-copy-mode if signatures match.

Ștefan:

- set loose-copy-mode on false if simpleCopy is true and test! To replicate, you should have two procedures with the same temporary table defined and used as Input-Output parameter. This way you will trigger simpleCopy as both tables will have the same DMO interface (with different multiplex). I would like to do an intermediate profiling round after this - notify after commit.
- set simpleCopy on false if the explicit signatures match. In this case, it is safe to do the same assumption like having the same DMO interface. I feel like this is a riskier change so needs better testing.

I think we should skip computing field mappings and other kind of time consuming operations if we can go the simple way. In practice, most of the copies happen before tables which are really similar (static or dynamic).

Please do some extensive testing here, especially with the tests from testcases/uast. It is important to test:

- extents (this is the most sensible part about fast-copy)
- append-mode (with / without fast-copy)
- replace-mode (with / without fast-copy)
- loose-mode (withhout options / with append / with replace-mode)

Check a large customer application. The change in #7404-6 in quite dependent on having a smooth and fast fast-copy support.

#11 - 06/21/2023 08:27 AM - Ștefan Roman

I added the first change, and I will start working/testing the second change.

Committed to 7404a revision 14629.

#12 - 06/22/2023 07:52 AM - Alexandru Lungu

- % Done changed from 60 to 80

With 7404a revision 14629 it is slightly better, but we are still 1% behind. However, I had really high variance on tests; my best testing iterations were even faster than my baseline before #7404.
Anyway, after you address #7404-10, we may want to consider improving FastCopyHelper. I will test together with #7389 as it consistently improves fast-copy with append.

At the end, I will do a full profiling on copy-temp-table to understand where it still working slow.

#13 - 06/22/2023 10:00 AM - Ștefan Roman

After letting simpleCopy be true when the explicit signatures matched, some custom small tests looked fine, but after running a large application the client crashed, and the error was thrown by TemporaryBuffer.copyAllRows, so I think we should try a different approach.

#14 - 06/23/2023 03:10 AM - Alexandru Lungu

Please do some forensics on that issue you have with the customer application. Do some debugging - extract the tables that don't match with fast copy-helper. Report back why isn't this working. My best guess is that we can adapt or improve FastCopyHelper to work with that failing case too. At worse, we can rule it out from canFastCopy.

As your testing environment is a customer application, please try to recreate the issue in a separate test-case and post it here.

#15 - 07/03/2023 03:33 AM - Alexandru Lungu

Stefan, please focus exclusively on this task. I am planning to merge 7404a in trunk asap; but the regression should be fixed first. Also, there may be some hidden performance drawbacks even after the regression is fixed, so there might be some extra work to be done after.

#16 - 07/04/2023 07:46 AM - Ștefan Roman

The problem was that the tables created dynamically had changed field names in SQL (field1, field2, ...) and the explicitSignature does not contain the names in SQL. I added a new signature, sqlSignature, that contains all the informations from explicitSignature + the fields names in SQL, and now simpleCopy will be true if the sqlSignature match. I tested with the customer application and worked fine.

Committed on 7404a, revision 14631.

#17 - 07/04/2023 08:43 AM - Alexandru Lungu

- % Done changed from 80 to 100

- Status changed from WIP to Review

Review to **7404a, revision 14631**

- The changes are OK, good job.

I will do a testing + profiling round. Keep you updated with the results.

#18 - 07/05/2023 04:36 AM - Alexandru Lungu

I done some testing and everything works fine. I only notices a slight improvement:

- The usual case is to match the DMO interface, so the signature check is not always needed. Thus, retrieve the srcBuf.getDmoInfo().getSignature().getSqlSignature() only when simpleCopy is false, giving it "a second chance". If it is already true due to srcBuf.getDMOImplementationClass().equals(dstBuf.getDMOImplementationClass()), avoid the signature retrieval. These calls are light, but as long as we can do it better fast, we shall do it immediately.
- EDIT: this is not related to this task, but I saw that the signature is eagerly computed on DMO meta creation. I think it is better to initialize it lazy. This way, we avoid computing the quickSignature, explicitSignature and schemaSignature to **all** DMO - we shall compute it only when necessary. Stefan, please check how other members of DmoMeta are retrieved and implement that for DMO signature. Also consider computing each signature lazily inside DmoSignature (compute it only when needed and cache).

#19 - 07/05/2023 07:33 AM - Alexandru Lungu

I tested with a large application and I got some interesting insights:

Simple copy	Append	Loose-copy	Extents	Before	Time	Count	Avg	Master Copy	Extent Copy	Before Copy	Before Extent Copy	Before Update Peer	Other
Without modifications													
No	No	No	No	No	29ms	264	0.10	23.07ms / 0.08	-	-	-	-	0.47ms / 0.001
Yes	No	No	No	No	29ms	313	0.09	21.94ms / 0.07	-	-	-	-	0.56ms / 0.001
Replace to append + looseCopy to false optimizations													
No	No	No	No	No	24ms	264	0.09						
Yes	No	No	No	No	23ms	313	0.07						
Yes	Yes	No	No	No	2.0ms	11	0.18						
No	Yes	Yes	No	Yes	63.5ms	74	0.85						
No	Yes	Yes	Yes	Yes	46.6ms	21	2.21						
All optimizations													
No	No	No	No	No	10ms	132	0.07	12.01ms / 0.09	-	-	-	-	0.22ms / 0.001
Yes	No	No	No	No	34ms	445	0.07	39.89ms / 0.08	-	-	-	-	0.90ms / 0.002
Yes	Yes	No	No	No	2.2ms	20	0.11	02.17ms / 0.10	-	-	-	-	0.04ms / 0.002
No	Yes	Yes	No	No	0.9ms	6	0.15	01.16ms / 0.19	-	-	-	-	0.01ms / 0.001
No	Yes	Yes	No	Yes	55.9ms	74	0.75	32.62ms / 0.44	-	24.42ms / 0.33	-	10.19ms / 0.13	5.17ms / 0.060
Yes	Yes	No	Yes	Yes	3.1ms	2	1.55	01.82ms / 0.91	0.15ms / 0.07	01.43ms / 0.71	0.17ms / 0.08	00.16ms / 0.07	0.17ms / 0.080
No	Yes	Yes	Yes	Yes	40.2ms	21	1.91	22.54ms / 1.07	2.68ms / 0.12	19.43ms / 0.92	2.19ms / 0.10	04.12ms / 0.19	1.65ms / 0.070

Of course, there are way more fast-copy operations happening after optimizations. I am not aware of the time spent out-side the fast-copy (so the one without optimization) - it may be interesting to find out the degree at which it was worth switching to fast-copy. Anyway, I still think there are some "hard nuts to crack": the scenarios that use before tables in fast-copy. These have times of 1/2 orders of magnitude higher, so we shall look into this topic next.

I think #7404 is already running out of its main course here (moving to fast-copy even if in replace mode). I will do my standard profiling routine and open a task on optimizing fast-copy of before-tables separately.

Stefan, please address the review of #7404 asap. If you get it in time, I may be able to catch it in my first profiling round - ideally.

#20 - 07/05/2023 08:31 AM - Ștefan Roman

I added the first change, where I retrieve sqlSignature only if simpleCopy is false, I will wait for another profiling.
Committed on 7404a, revision 14632

#21 - 07/06/2023 08:40 AM - Alexandru Lungu

Created #7448 and #7449 to further optimize the temp-table (fast-)copy process.

#22 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Bug #7448: Optimize FWD-H2 ValueTimestampTimeZone and maybe avoid caching added

#23 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Bug #7449: unqualified _File buffer must target the first database in the connection list added

#24 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to deleted (Bug #7449: unqualified _File buffer must target the first database in the connection list)

#25 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to deleted (Bug #7448: Optimize FWD-H2 ValueTimestampTimeZone and maybe avoid caching)

#26 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Bug #7488: Slow fast-copy with before tables in H2 added

#27 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Bug #7489: Lazily initialize DMO signature added

#28 - 07/07/2023 03:14 AM - Alexandru Lungu

The current performance degraded even more to +0.6%. This may be due to the slow fast-copy we do. I guess most of the copies actually move only some records (1/2) and thus the overhead of doing INSERT INTO SELECT FROM is higher. The statistic in #7404-19 is self-explanatory.

However, I think we moved the things in the right direction here. This way, we can optimize before table copies per-se and get a visible improvement. Before the changes here, optimizing replace-mode was mostly impossible as it required doing a one-by-one kind of job. Right now, we should just take care of how we actually implement fast-copy for before.

In this state, 7404a can't be merged. Stefan, please continue the work on #7489 and #7488 and dump the changes there in 7404a branch.

#29 - 07/07/2023 03:17 AM - Alexandru Lungu

Stefan, please add these two sub-tasks (#7489 and #7488) to the H2 performance Wiki asap.

#30 - 07/07/2023 06:16 AM - Alexandru Lungu

Rebased 7404a to latest trunk. 7404a is now at rev. 14651.

#31 - 07/10/2023 05:13 AM - Alexandru Lungu

I added a break-down analysis of each operation done by fast-copy in #7404-19. Therefore, note that the cumulative times may mismatch. I quite lost the intermediate configuration I had for the middle part, so I skipped recomputing the break-down there. The format is: Time / Avg

#32 - 08/21/2023 10:01 AM - Radu Apetrii

While testing something for #7488, I noticed a regression on a test with copy-temp-table with REPLACE, introduced in 7404a, rev. 14650.
The test consists of a copy-temp-table in replace mode, the desire being to actually replace a record in the destination table.

```
define temp-table tt no-undo field f1 as int field f2 as int index idx1 is unique f1.
define temp-table tt2 no-undo field f1 as int field f2 as int index idx2 is unique f1.

create tt. tt.f1 = 1. tt.f2 = 1.
create tt. tt.f1 = 2. tt.f2 = 3.

create tt2. tt2.f1 = 2. tt2.f2 = 2.

buffer tt2:handle:copy-temp-table(buffer tt:handle, false, true).

for each tt2:
    message tt2.f1 tt2.f2.
end.
```

In 4GL, the test shows two records, while in FWD an error is raised.

If this won't be solved by the time I finish #7488 and there are no objections on this, I'll investigate this matter myself (since I'm working on something related). In addition to that, there have been added quite a lot of changes to 7404a and it could be difficult to integrate some changes.

#33 - 08/22/2023 05:17 AM - Alexandru Lungu

- Assignee changed from Ștefan Roman to Radu Apetrii

Radu, this task was mostly fixed, but it is still open for review. As you do a major refactoring of FastCopyHelper, please use the feedback here to do your testing.

#34 - 08/23/2023 08:31 AM - Radu Apetrii

Radu Apetrii wrote:

> While testing something for #7488, I noticed a regression on a test with copy-temp-table with REPLACE.

The problem was that when the source record was copied into the destination record, its primary key (a.k.a. BaseRecord.id) was also copied. This lead to a mismatch of primary keys when attempting to store the record in the corresponding table. The commit that fixes this issue is on 7404a, rev. 14657.

#35 - 09/06/2023 08:53 AM - Constantin Asofiei

I'll do a round of ETF tests.

In the mean time, please fix the history numbers - one per file, for the very first log message added.

#36 - 09/06/2023 09:15 AM - Constantin Asofiei

Alexandru, do you have a round of performance testing with 7156b + 7404a? I ask because #7404-28 notes a performance degradation.

#37 - 09/06/2023 09:34 AM - Alexandru Lungu

Fixed the history numbers. I tested today the performance and I had 7 runs: some with 0.9% - 1.5% perf. improvement and some runs with 0.8% - 0.9% perf. decrease. However, I get an average of -0.2% so I am quite content with that. Finally, the average is in the right direction.

#38 - 09/06/2023 10:58 AM - Constantin Asofiei

Constantin Asofiei wrote:

> I'll do a round of ETF tests.

ETF tests is OK. Actually, it fixes a test which previously failed.

You can merge to trunk now, after rebase.

#39 - 09/06/2023 12:28 PM - Constantin Asofiei

Merged is postponed for tomorrow - please coordinate with Greg before queuing for merge.

#40 - 09/07/2023 02:47 AM - Alexandru Lungu

OK.

#41 - 09/07/2023 06:54 AM - Greg Shah

If you are ready, you can go ahead with the merge now.

#42 - 09/07/2023 08:02 AM - Alexandru Lungu

Starting the rebase; I will notify you when ready.

#43 - 09/07/2023 09:04 AM - Alexandru Lungu

- Status changed from Review to Test

Committed 7404a to trunk as rev. 14730.

#44 - 01/09/2024 08:04 AM - Alexandru Lungu

This can be closed.

#45 - 01/09/2024 08:16 AM - Greg Shah

- Status changed from Test to Closed