Database - Bug #7488

Slow fast-copy with before tables in H2

07/06/2023 06:21 AM - Alexandru Lungu

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

Description

Related issues:

Related to Database - Feature #7404: Trasform replace-mode into append-mode w...	**Closed**

History

#1 - 07/06/2023 06:23 AM - Alexandru Lungu

- Assignee set to Ștefan Roman

- Status changed from New to WIP

This is mostly related to #7404-19. After several optimizations, the most time consuming operations of fast-copy are the ones that use before tables. I suspect this is because FastCopyHelper.updatePeerRecords is doing a "one-by-one" iteration to update peer-id.

For this task:

- Do a test with fast-copy without before tables.
- Do a test with fast-copy and before tables.
- Compare performance and bottle-necks; confirm FastCopyHelper.updatePeerRecords is dragging back the performance here
- Search for an optimization.

#2 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Feature #7404: Trasform replace-mode into append-mode when target table is empty added

#3 - 07/18/2023 03:44 AM - Ștefan Roman

After some investigations, FastCopyHelper.updatePeerRecords is not exactly the bottle-neck. When using fast-copy with before tables, most time is spend in FastCopyHelper.copyMasterTable because it also have to do the mapping. One optimization we can make is set append flag to false if the destination tables are empty (both master and before) since we don`t need to do the mapping anymore (the signature is checked before doing fast-copy). I will keep looking for other improvements.

#4 - 07/18/2023 07:23 AM - Alexandru Lungu

Alright; please go ahead with a commit in this sense to 7404a

#5 - 07/18/2023 07:54 AM - Greg Shah

When we copy between any 2 given DMOs, do we save and reuse all the work we do to figure out the "copying plan"?

#6 - 07/18/2023 08:05 AM - Alexandru Lungu

I like the "copying plan" terminology :) This is accurate. We keep a database level mapping on the keys (source to destination) for master table and eventually before table. The plan follows:

- compute the copying plan (pk mapping), unless it can be delayed to a later stage
- do the copy for master table (using the pk mapping if exists)
- do the copy for normalized extent tables (using the pk mapping or generating one if it was delayed)
- do the copying plan (pk mapping) for before tables, unless it can be delayed to a later stage
- do the copy for before table (using the pk mapping if exists)
- do the copy for normalized extents for before tables (using the pk mapping or generating one if it was delayed)
- reset the updatePeerRecords based on the existing pk mapping if any or generate the required pk mappings
- other work - for example, if we were asked to retrieve the destination id of a certain source id after the copy process

Note that simple copies don't require any mapping (no normalized extents or before tables). Some of the may require only one mapping (no before tables). Other may require more work (extents + before).
Certain copy processes are "extra complex" if append is used, as it requires some implicit unique checks (so there is an INSERT INTO [...] SELECT FROM [...] WHERE [..index is no violated..].
Even more complexity is added by loose-copy as we need to remap the columns from source to destination.

> When we copy between any 2 given DMOs, do we save and reuse all the work we do to figure out the "copying plan"?

Now that I have read your comment a second time .... do you mean if we reuse the copying plan across copy-temp-table statements? So if two tables are part of a copy process, the same "plan" is reused? Well, this is not a performance concern from my POV; the only consuming tasks here are the SQL queries, which are prepared and cached anyways. Even updatePeerRecords is just a bunch of update statements. Also, the internal "pk mapping" we do can't ever be reused.

#7 - 07/18/2023 08:18 AM - Greg Shah

> do you mean if we reuse the copying plan across copy-temp-table statements? So if two tables are part of a copy process, the same "plan" is reused?

Yes, that is exactly what I'm asking.

> Well, this is not a performance concern from my POV;

OK, if there is not any appreciable work that can be saved then we don't need to think about caching it.

#8 - 07/18/2023 08:45 AM - Greg Shah

Isn't there some string comparison overhead in calculating the field name mappings?

#9 - 07/18/2023 10:17 AM - Alexandru Lungu

Well, there are some bits of code using strings. Even the signature match does some string comparisons. The SQL composition itself may be costly (as we concatenate all field names), but all parts of any single SQL is done using StringBuilder. I know this also implies some string work, but I am not sure at what extent is this a problem. We already have #7389 and #7351 which introduced (for example) * wildcard instead of listing all properties.

I agree there is still some work in this direction. I am actively tracking the fast-copy statements from a profiling POC and trying to understand which optimizations can be done. First phase was to hugely decrease the size of the statements. I guess the second phase is optimizing this string work as you said.

#10 - 07/21/2023 08:58 AM - Ştefan Roman

I added this first change. I tested with a small test and a client app and it works fine. Committed on 7404a, revision 14652.

#11 - 07/25/2023 04:15 AM - Alexandru Lungu

Review of 7404a, revision 14652.

- The lines you changed now exceed 110 characters. Consider extracting the last parameter into noMapping.
- I am redoing my tests from #7404-19 - lets see if there are changes.

#12 - 07/25/2023 06:37 AM - Alexandru Lungu

Simple copy	Append	Loose-copy	Extents	Before	Time	Count	Avg	Master Copy	Extent Copy	Before Copy	Before Extent Copy	Before Update Peer	Other
7404a / rev. 14651													
No	No	No	No	No	13ms	152	0.08	13.60ms / 0.08	-	-	-	-	0.17ms / 0.001
Yes	No	No	No	No	39ms	461	0.08	38.78ms / 0.08	-	-	-	-	0.72ms / 0.001
Yes	Yes	No	No	No	2.6ms	24	0.10	02.66ms / 0.10	-	-	-	-	0.02ms / 0.000
No	**Yes**	Yes	No	No	0.8ms	6	0.16	00.85ms / 0.14	-	-	-	-	0.01ms / 0.001
No	**Yes**	Yes	No	Yes	36.4ms	74	0.49	09.27ms / 0.12	-	6.51ms / 0.08	-	16.53ms / 0.22	4.27ms / 0.057
Yes	**Yes**	No	Yes	Yes	0.9ms	2	0.45	00.16ms / 0.08	00.28ms / 0.14	0.20ms / 0.10	0.15ms / 0.06	00.07ms / 0.03	0.11ms / 0.055
No	**Yes**	Yes	Yes	Yes	22.3ms	21	1.06	04.41ms / 0.21	05.24ms / 0.12	4.44ms / 0.24	2.68ms / 0.12	03.61ms / 0.17	1.98ms / 0.094

The changes optimize the append cases quite well.

- Master Copy times are now lower on average, because there were many append over empty tables, so this could be optimized (delay easier PK mapping)
- Extent Copy (for master and before) is a bit slower as the PK mapping is delayed until this point now.
- Before Copy is **way more efficient**. This is because it doesn't do any eager PK mapping if there are no append and the delayed PK mapping is faster.
- Update Peer is just a bit slower as most delayed PK mappings should be built here.

The total time now for all copy operations is ~115ms, while before we faced ~146.3ms. There is no obvious slow-down here - all operations complete in <0.25ms.

For the more complex operations, I guess we can even batch the SQL we are using, right? However, I don't think there is much benefit considering that the underlying database is in-memory.

I will check next #7488-7: are there any copy operations that repeat (have the same source-destination pair)?

Also, I think the following is a typo in FastCopyHelper.executeCopy:

```
            dstBuf.invalidateFFCache(0);
            if (srcB4Buf != null)
            {
               srcB4Buf.invalidateFFCache(0);
            }
```

I guess srcB4Buf should be replaced with dstB4Buf.

#13 - 07/25/2023 06:43 AM - Alexandru Lungu

- % Done changed from 0 to 50

Last minute notice: updatePeerRecords doesn't use prepared statements, but generates plain SQL statements. **Stefan**, please refactor the code there so prepared statements are used for _PEER_ROWID updates.

#14 - 07/26/2023 05:21 AM - Alexandru Lungu

Alexandru Lungu wrote:

> I will check next #7488-7: are there any copy operations that repeat (have the same source-destination pair)?
>
> Also, I think the following is a typo in FastCopyHelper.executeCopy:
> [...]
>
> I guess srcB4Buf should be replaced with dstB4Buf.

Nevermind; this can be removed entirely. We already invalidate the ffcache once we exit the FastCopyHelper.executeCopy in TemporaryBuffer.

#15 - 07/26/2023 07:53 AM - Alexandru Lungu

Committed 7404a/rev. 14653. This resets the loose-copy flag if the loose property matching is the same as the same as the simple-copy one (all properties are included in the same order). This way, I got slightly faster table copies due to the use of the * wild-mark, both in source and destination. Stefan, please update your 7404a.

I have only 9 copy operations in my POC that aren't covered by fast-copy. There take 3.1ms in total, so we can skip the effort of getting more SQL in fast-copy mode.

I've done some investigation and there seems to be many copy operations that are repeated, so we can re-use the previously computed "bundle of SQLs" to do the copy. **However**, most of the repeating operations were trivial anyway - the most consuming ones are between static tables and dynamic tables - and these don't quite repeat.

#16 - 07/26/2023 08:10 AM - Greg Shah

> the most consuming ones are between static tables and dynamic tables - and these don't quite repeat.

It is a common pattern the in 4GL to do something like this:

- create a new dynamic TT LIKE a static non-temp database table
- copy records from the database table into the new temp-table
- process the temp-table
- generate output from the tt or copy the tt data back to the database table

Perhaps they don't repeat because each time through a new dynamic tt is created (but it is really the same table every time).

#17 - 07/26/2023 09:11 AM - Ștefan Roman

I fixed the line longer than 110 characters and removed the invalidation block. I changed updatePeerRecords so now it is using Persistence.executeSQLBatch with PreparedStatement.
Committed on 7404a, revision 14654.

#18 - 07/27/2023 09:03 AM - Alexandru Lungu

Greg Shah wrote:

> Perhaps they don't repeat because each time through a new dynamic tt is created (but it is really the same table every time).

This is right. Thus, the SQLs can't be cached / prepared and resued mostly because the dynamic tables will have different names (dtt1, dtt2, etc.). However, most of the SQLs match (as the fields as named with field1, field2, etc.). Basically, only the table name mismatches, making the statements not preparable / cachable.

I am still looking forward to cache such statements even for the simple static-to-static cases.

#19 - 07/27/2023 09:16 AM - Greg Shah

Do we really need the table names to be different at the database level? Why not reuse the same table and DMOs? It is not like the tables can be referenced by name in the source code. It seems like the temp-table names don't matter.

#20 - 08/03/2023 03:26 AM - Alexandru Lungu

- Assignee changed from Ștefan Roman to Radu Apetrii

Radu, there is some work in #7404 regarding the moving of most of the copy-temp-table operations to fast-copy using heuristics (e.g. copy in REPLACE mode with empty destination is equivalent to APPEND mode, which is faster). We reached >90% of the operations to be "fast" on a customer POC. This is good, but it seems that our current fast-copy is in fact slower for some cases (see #7488-12). **We need a refactoring of FastCopyHelper!**

To start with, we need FastCopyHelper to be a cachable object based on the source and destination DMO + before source and destination DMO + copy mode bitmask (NORMAL, APPEND, REPLACE, LOOSE-MODE). This will need some refactoring, of course. The final goal is to reuse FastCopyHelper with all of its underlying SQL. Basically, we need to build a "copy bundle" that is an SQL batch which after execution will do the table copy. Right now, all SQL are generated all over again for each copy. Try to find the best solution without altering the current behavior. Please use 7404a for development, as it already has the latest improvements there.

Important!: some of the optimizations are contextual (depending if the target is empty or not). Therefore, inside FastCopyHelper, there are changes in the copy mode (e.g. if the target is empty, don't do APPEND, but simple copy). Such mode changes should be tackled in TemporaryBuffer.copyAllRows, so that we pick up the right FastCopyHelper in the end, without implicit mode changes. Otherwise, we might cache the wrong SQL just because we had a context - **no**, we should cache FastCopyHelper not dependent upon any context (e.g. if the target is empty or not).

Please mark this as one of your top priorities.

#21 - 08/17/2023 06:26 AM - Radu Apetrii

I've done some refactoring to FastCopyHelper:

- Added a cache that stores the instances of FastCopyHelper based on a FastCopyKey.
- A FastCopyKey is generated with the help of the class from source buffer, destination buffer, source before buffer, destination before buffer and an integer denoting a bitmask for the copy mode (APPEND, REPLACE, etc.).
- Added FastCopyBuilder which allows the creation of a FastCopyHelper.
- Rewrote the static methods so that they are no longer static (apart from the ones that use the cache).
- In TemporaryBuffer, instead of having multiple method calls to FastCopyHelper, the process has been refactored so that there is one call (via FastCopyHelper.tryExecuteCopy).

The commit is on 7404a, rev. 14655.
After some performance testing, an improvement is definitely visible. However, I believe there is still room for improvement.
Here are some results (note that a positive value means improvement):

Table size/populate	Test-loose improvement	Test-index improvement	Test-fields improvement	Test-extent improvement
small/small	11.34%	5.49%	11.34%	6.35%
small/medium	2.67%	2.16%	3.21%	0.09%
small/large	1.77%	not tested	3.11%	not tested
medium/small	5.93%	0.62%	1.36%	7.79%
medium/medium	1.89%	2.28%	0.66%	1.05%
large/small	6.57%	3.51%	2.35%	3.39%

#22 - 08/17/2023 11:15 AM - Alexandru Lungu

Review of 7404a

This is a good start, but it is still far from ideal at this point. Radu, please consider doing more radical changes to get this state-of-the-art.

- destRecId = new rowid(); destRecId.assign(copyResult); doesn't have sense. destRecId comes as a parameter, so reassigning here won't help! It is caller's responsibility to pass down a non-null destRecId if srcRecId is set. You can pass down null, but in this case just skip the assignment. Also, I don't think dstQueried is required here. Finally, move srcRecId next to destRecId parameter.
- I think you can create a FastCopyKey based on a FastCopyBuilder directly (I mean using a constructor with a single parameter, a FastCopyBuilder).
- fastCopyCache shouldn't be a HashMap because it can leak. There is not a finite number of DMO through-out the execution (as there are dynamically generated DMOs). Transform it into a LRUCache. Please look into Danut changes on CacheManager.createMapCache. Set the size on 256.
- dstIsTableEmpty, srcB4IsTableEmpty and dstB4IsTableEmpty don't look right. When you initialize the fast-copy, these will be set according to the **current** state of the application. As these are not part of the cache key, you may end up hitting the cache with non-empty tables, but with these flags set on true. **This might cause regressions**. All flags from the FastCopyHelper should be conceptually immutable. This is the case for most of them (srcMeta, srcDmoInterface, srcDialect, srcTableName, etc.). A trivial solution is to include the "empty" flags into the cache keys.
- Please double check Persistence and if it can be cached inside FastCopyHelper and reused. Hopefully it won't be subject to any mutability.
- There might be some concurrency issues with fastCopyCache. I think this should have synchronized access. AFAIK, LRUCache is not synchronized:
  - Note that TemporaryBuffer.Context is a class which is per-session, so you can move the cache there (recommended)
  - Create a ContextLocal instance in FastCopyHelper.
  - Do explicit synchronization of the cache - this means that the cache is shared. This might be quite tragic as long as you need to store Persistence inside (or other per-session resources).
- The overhead of storing each SQL in separate variables is huge. I mean, it is functional, but far from maintainable. The issue here is that you keep 13 string members, but only some of them may be non-null. I would suggest for a start to:
  - Consider splitting the code for master from the one for before. Basically, imagine that there are two independent copy operations. The only point where they converge is updatePeerRecords. You can generate one FastCopyHelper for master and one for before. After you execute both, just call updatePeerRecords as a static method, passing down the FastCopyHelper / FastCopyContext instances and "pick up" whatever you need from there. Having everything duplicated (for master and before) is something I want to get rid of asap. This way you can make light-weight cache keys, less properties, etc.
  - All methods that execute SQL (e.g. safeTableCopyNoMapping) can be refactored to only append the SQL into a "bundle" (ArrayList with pairs of query string and Object[] args). Note that args should be dynamically computed and not stored! That is, instead of Object[] args, you will need Function<FastCopyContext, Object[]> that will use the fast-copy helper/context to detect the proper arguments at the right time.
  - In the end, each FastCopyHelper will basically compute a FastCopyBundle. So, if the bundle is not computed, compute it. If it is computed, run it using the FastCopyContext.
- This is something I wanted to add a very time ago, but didn't had a clean code to do it right. Please set a save-point before attempting to run the bundle. In case anything occurs, consider reverting the whole bundle. This avoids partial fast-copy attempts.

This is a long note. Please feel free to discuss further anything from above.

The goal here is:

- For one copy attempt (master or before)
  - Extract an existing FastCopyHelper from the cache if exists or create a new one.
  - Compute the FastCopyBundle if exists or use it directly.
  - Combine the FastCopyBundle and FastCopyContext to do the execution. Basically, iterate one-by one: the SQL query, compute the arguments and run.
- If we had both master and before, call updatePeerRecords.

Minor:

- srcBuf.getDmoInfo() is called several times, even if there is already a srcMeta. This also applies for dest and before.
- I think you better rename FastCopyBuilder into FastCopyContext. This basically means the run-time context in which you use the FastCopyHelper. **I want to stress out the importance of splitting the immutable data from the mutable data. Keep immutable in FastCopyHelper, move mutable in FastCopyContext. Use the FastCopyHelper with the right context at the right time**

For now, you can omit destRecId as it can be easy to handle at the very end.

#23 - 08/23/2023 06:50 AM - Radu Apetrii

I added some more changes to FastCopyHelper on 7404a, rev. 14656:

- Merged the master/before methods, meaning that instead of having, for example, a getMasterMapping and a getBeforeMapping, there is now only one method, getMapping.
- Moved the fastCopyCache to TemporaryBuffer.Context. This implies that this cache is now per-session instead of being static.
- Created a FastCopyBundle insinde FastCopyHelper which does the following:
  - It stores the SQLs that should be executed instead of having them executed on the spot.
  - It executes the SQLs that have been stored at the end of the copying process.
  - In case the fastCopyCache is hit, then the bundle is ready to go and be executed instead of having to wait for the SQLs to be computed.
- Added a savepoint in case the execution of the SQLs fail and a rollback is necessary.
- Other not-so-major changes (suggested in #7488-22), such as changed the fastCopyCache from a HashMap to a LRUCache, changed the name of FastCopyBuilder to FastCopyContext, etc.

I ran several tests and got no error. Performance wise, I still get a decent improvement, but I am yet to test with a large customer application.

#24 - 08/25/2023 03:57 AM - Alexandru Lungu

- % Done changed from 50 to 70

Review of 7404a, rev. 14656

The changes are quite good now and look pretty stable. However, there are still some improvements that can be made, especially on the performance side. Radu, prepare this branch for a final review / testing / profiling. Thus, do a double/triple check and testing. Consider debugging into the solution to ensure it does the right job.

- Please dig in a bit into the append / replace / loose-copy cases. If you have append |-> true replace |-> true, but the destination is empty, this will cause append |-> true replace |-> false. However, this check is done in tryExecuteCopy, which is quite late. This means the cache will have both append |-> true replace |-> true **and** append |-> true replace |-> false with the same underlying bundle. You shall do the isTableDefinitelyEmpty check in TemporaryBuffer and **alter** the append / replace flags **before** doing the actual cache look-up. In essence, tryExecuteCopy looks like an overhead here.
- The append flag is propagated through almost all methods - is this still required? Can't we use the class-level append?
- You can eliminate the *Sql class members, as the SQLs are stored in the bundle anyway. No need for a two-level caching here.
- for (int i = 1; i < argSize; i++) arguments[i] = args.get(i); is this safe? It works as it start from 1, so it avoids context related arguments. However, it looks quite easy to break. Consider leaving args only for non-contextual arguments and append contextual arguments (like fastCopyContext.dstBuf.getMultiplexID()) separately (outside args).

Minor:

- Replace Context ctx = context.get(); with Context ctx = local;, or use local directly. Once you are in a temporary buffer (non-static method), you have the context already resolved inside local. Using get will do some thread-local type of work to identify the current context; this is not required.
- Add javadoc to Context.getFastCopyHelper.
- FastCopyHelper.finishCopy can now return destRecId, instead of assigning it.
- Please attempt before copy only if before exists. Therefore, avoid the whole cache look-up and copy attempt if before are not enabled.
- In FastCopyHelper.updatePeerRecords, you should use beforeCopyHelper.srcPersistence.executeSQLBatch(sql, supp), instead of the first masterCopyHelper.srcPersistence.executeSQLBatch(sql, supp).
- In executeSqls, you can rethrow the same exception.
- FastCopyContext.pk is quite confusing. Is this the first pk used for the destination table?
- You can inline addBundleElement with new FastCopyBundleElement, or even do a sugar-method addBundleElement(String sql, Function<FastCopyContext, Object[]> args)
- For lambda function, you need to have the open brace ({) on a separate line.
- generateArgs in copyExtentFields can simply return args. I don't think a copy of the Object array is actually required here.
- You can make FastCopyKey of type byte. It suits better the bit masks with a low number of bits.
- FastCopyBundle can be a static class (?)
- if (element instanceof FastCopyBundlePKElement) is not required, just call execute. The overriding will do its job - as long as you make execute non-private.
- extends clause should be on a separate line.

#25 - 08/28/2023 06:03 AM - Radu Apetrii

I've applied the suggestions from the last post (except one) and committed to 7404a, rev. 14658.

Alexandru Lungu wrote:

- Replace Context ctx = context.get(); with Context ctx = local;, or use local directly. Once you are in a temporary buffer (non-static method), you have the context already resolved inside local. Using get will do some thread-local type of work to identify the current context; this is not required.

The problem with this is that TemporaryBuffer.copyAllRows is a **static** method, so I don't have access to local, which is **non-static**. I tried some workarounds too see if I could gain access to it, but I couldn't get rid of Context ctx = context.get();.

#26 - 08/28/2023 08:07 AM - Alexandru Lungu

Radu Apetrii wrote:

> The problem with this is that TemporaryBuffer.copyAllRows is a **static** method, so I don't have access to local, which is **non-static**. I tried some workarounds too see if I could gain access to it, but I couldn't get rid of Context ctx = context.get();.

You can still extract the context from one of the buffers (srcBuf.local). In theory, all 4 buffers (source/destination, master/before) should have the same local context.

#27 - 08/28/2023 08:17 AM - Radu Apetrii

Alexandru Lungu wrote:

> Radu Apetrii wrote:
>
> > The problem with this is that TemporaryBuffer.copyAllRows is a **static** method, so I don't have access to local, which is **non-static**. I tried some workarounds too see if I could gain access to it, but I couldn't get rid of Context ctx = context.get();.
>
> You can still extract the context from one of the buffers (srcBuf.local). In theory, all 4 buffers (source/destination, master/before) should have the same local context.

Right, I didn't think of that, thank you. I swapped context.get() with srcBuf.local and committed to 7044a, rev. 14659. I did not encounter issues when testing.

#28 - 08/29/2023 10:45 AM - Alexandru Lungu

- % Done changed from 70 to 90

Review of 7404a

- You don't actually need FastCopyContext.newInstance - you can call the constructor directly.
- You can cache canFastCopy result. If you ever attempt to do a copy that you know it can't be done fast, then you can simply short-circuit canFastCopy. Also, if you know once that the copy can be done, no need to double check the signatures. Note the javadoc is not right for return - it shouldn't mention the return type
- There is quite some logic around isBefore inside executeCopy. Can you summarize it here - precisely why is it needed? What are the differences between master and before copies?
  - I am worried that if (fastCopyBundle.isBundleComputed()), this returns true only for before tables, otherwise it returns false.
  - Also, copyExtent is true only for master copies. Note that before tables can have extents too that need to be copied!
  - The hasBeforeRecords && fastCopyBundle.isBundleComputed() conditional is checked twice. The second occurrence can be removed.
- I don't see the point of tryExecuteCopy anymore. You shall replace success with canFastCopy and call executeCopy for master directly in TemporaryBuffer.
- do a double-check for the javadoc. There is a lot of code changed here, so we need to make sure the javadoc is not referring to the old (un-cached) approach.

#29 - 08/30/2023 04:06 AM - Radu Apetrii

Alexandru Lungu wrote:

> - There is quite some logic around isBefore inside executeCopy. Can you summarize it here - precisely why is it needed? What are the differences between master and before copies?

isBefore was introduced to serve three purposes:

- When assembleUidMapping is called, the program isn't aware if the copy process is happening for master of before, thus, it has no idea if the parameter for the function is supposed to be BEFORE__PK__MAPPING or MASTER__PK__MAPPING. **I guess this can be refactored so that there is only one mapping stored** which is determined in the initialize method.
- In copyTable, a master copy can be executed through either safeTableCopyNoMapping, safeTableCopy, or copyTable. For a before copy, though, there are only two methods available: safeTableCopy and copyTable. **The role of isBefore was to eliminate the possibility of a before copy to execute safeTableCopyNoMapping** taking into account the fact that the parameters for copyTable could have the same values for both master and before.
- As of 7404a, rev. 14658-14659, this reason kinda lost its point, because finishCopy got separated from clear. Before those changes, finishCopy was handling both the execution of SQLs for the master copy and the clear part (with drop table ...). Since I didn't have access to the before copy when I was in masterCopyHelper.finishCopy in order to execute the before SQLs between the master SQLs and the clear part, I reorganized the code a bit:
  - For the master copy:
    - Enter executeCopy.
    - If the bundle is already computed, **then quickly exit**, because the SQLs are already generated and will be executed in finishCopy.
    - If the bundle is not computed, then execute copyTable and generate the SQLs which will be executed later.
  - For the before copy:
    - Enter executeCopy.
    - If the bundle is already computed, **then execute the SQLs right away** because it won't have a chance later to do that.
    - If the bundle is not computed, execute copyTable to generate the SQLs and then execute the SQLs.

Because finishCopy got separated from clear, I think I can mimic the behavior of the master copy for the before process, meaning that executeCopy won't behave differently depending on the type of copy. This seems more logical and less complicated.

The only relevant purpose at the moment for isBefore would be bullet 2 from the list above. If you wish, I can try to get rid of it for good. Also, I will address the other points from the #7488-28 and I will keep you up to date.

Note: In copyTable I will have to double-check the conditions because it seems that before copy doesn't have access to safeTableCopy at the moment.

#30 - 08/30/2023 04:19 AM - Alexandru Lungu

Radu Apetrii wrote:

> The only relevant purpose at the moment for isBefore would be bullet 2 from the list above. If you wish, I can try to get rid of it for good. Also, I will address the other points from the #7488-28 and I will keep you up to date.

That will be great. The only thing to avoid for before is to do safeTableCopyNoMapping, because mappings are always needed when you do before copies. You can keep isBefore in the cache key and fast-copy helper to help you out with this. But make sure the use of this flag is limited only to this scenario.

#31 - 08/30/2023 08:09 AM - Radu Apetrii

I applied the bullets from #7488-28 and committed to 7404a, rev. 14660. In addition to that, I also merged the two processProperties functions since one of them was just calling the other one without any consequences.

There were no issues found while testing, but I'm still waiting for the database import to finish in order to test with a large customer application.

#32 - 08/31/2023 09:58 AM - Alexandru Lungu

The changes are very close to state-of-the-art in 7404a. Very good job, Radu!

- Please rename PK__MAPPING into pkMappingName. This is not static anymore, so it is not a constant per-se.
- Is there any reason why canFastCopy should be cached when validTempTableBulkCopy returns false?
- I would expect that executeCopy would actually **execute** the copy. In your case, the finishCopy actually executed. Can we make executeCopy execute the bundle if computed and eventually return the dstRecId? Is the separation between executeCopy and finishCopy needed?
- We can avoid isPKComputed by considering dstPK as null when not computed.
- **Danut, did you had some correctness tests with fast-copy at some point?** Can you share them here?

#33 - 09/01/2023 02:10 AM - Dănuț Filimon

- File ctt-tests.zip added

Alexandru Lungu wrote:

> Danut, did you had some correctness tests with fast-copy at some point? Can you share them here?

I attached my tests that I used in #7389. Note that I already shared this test suite with Radu at his request, so I don't know if it's going to help.

#34 - 09/04/2023 04:00 AM - Radu Apetrii

Dănuț Filimon wrote:

> Alexandru Lungu wrote:
>
> > Danut, did you had some correctness tests with fast-copy at some point? Can you share them here?
>
> I attached my tests that I used in #7389. Note that I already shared this test suite with Radu at his request, so I don't know if it's going to help.

At the time of posting the solution (#7488-31), the program was already tested with Danut's examples, and there were no issues. But thank you for the intervention.

I committed to 7404a, rev.14661 the changes suggested in #7488-32. I did not encounter problems when applying them.

#35 - 09/05/2023 05:19 AM - Alexandru Lungu

Radu, in 7404a, the immutable and mutable data still not independent from each other:

- In FastCopyHelper.executeCopy, noMapping depends on the context (as it depends both on hasBeforeRecords and hasSourceId which is contextual). Note that hasBeforeRecords depends on isTableDefinitelyEmpty. Therefore, you can compute different kind of bundles depending on the state.
  - noMapping should be included into the key, to distinguish between copies requiring mapping and the ones that don't.

I made this change in 7404a/rev. 14662. Radu, please review.

#36 - 09/05/2023 05:41 AM - Radu Apetrii

Alexandru Lungu wrote:

> I made this change in 7404a/rev. 14662. Radu, please review.

The changes make sense, it was important to fully separate the contextual variables from the non-contextual.
However, while scrolling through the class, I **accidentally** found a line that has over 110 characters: in FastCopyHelper.clear, where you added if exists to the SQL. Apart from that, it looks great. I'm looking forward to the profiling results.

#37 - 09/05/2023 10:51 AM - Alexandru Lungu

- % Done changed from 90 to 100

- Status changed from WIP to Review

I've done a second commit on 7404a removing more mutable data and making the solution more stable (rev. 14663). Now I am not facing regressions anymore:

- clear is dependent upon context (if it created a mapping or not on its way)
- remove append if possible in TemporaryBuffer, not in FastCopyHelper
- removed srcToDstPks which was context dependent.
- removed copyMeta as it was redundant
- removed some SQL query caches as they were redundant.
- removed replace mode from the fast-copy key as fast-copy doesn't support replace mode. It is resolved before the fast-copy is actually used.

I will start running my regression tests as they are stable enough now.

> However, while scrolling through the class, I accidentally found a line that has over 110 characters: in FastCopyHelper.clear, where you added if exists to the SQL. Apart from that, it looks great. I'm looking forward to the profiling results.

I removed the if exists and kept the server-side state in generatesPkMapping. This is no longer a problem.

#38 - 09/11/2023 05:55 AM - Alexandru Lungu

- Status changed from Review to Test

Committed 7404a to trunk as rev. 14730.

Files

ctt-tests.zip	7.86 KB	09/01/2023	Dănuț Filimon