Database - Feature #7037

remove dirty share manager

01/12/2023 01:53 PM - Eric Faulhaber

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

Description

Related issues:

Related to Database - Support #7963: disable dirty share database and test ex...	**New**
Related to Database - Feature #7964: investigate the benefits of disabling cr...	**New**

History

#1 - 01/12/2023 02:30 PM - Eric Faulhaber

The dirty share manager originally was added to solve a problem with a particular use case in one customer's application. The code was written before the 4GL supported sequences. To emulate sequence atomicity, the code took advantage of a transaction isolation quirk in the 4GL, whereby uncommitted updates to indices are visible between sessions.

The dirty database was added to FWD to emulate this quirk to allow that application to work as written. However, over time, the implementation evolved to also be a way to allow FIND statements within a session to return records which are newly created and not yet fully updated (and thus potentially invalid w.r.t. unique indices). In FWD terms, these records have not yet been validated and flushed to the database, but because the 4GL allows them to be found by other buffers in the same session at this early stage, the dirty database was re-purposed to emulate this behavior. To accomplish this, unique indices in the primary database are mirrored in the dirty database, but the unique constraint is removed, to allow these "early" records to be stored without error in the dirty database.

The dirty share implementation may be integrated with other functions of the persistence runtime, but it's gotten complex enough that I can't fully remember where.

The implementation has flaws:

- it adds significant overhead in terms of CPU and memory;
- it only works with FIND statements (we did not integrate this with other record retrieval statements, due to the complexity of implementation and performance overhead);
- it does not work properly when a FIND statement affects more than one table (e.g., with a nested CAN-FIND on another table);
- it is complex to maintain.

We want to get rid of it. We can address the original requirement (the cross-session transaction isolation quirk) in a different way, most likely with changes to 4GL application code. An actual dependency on the isolation quirk is rare; we have found it used intentionally only on this one occasion. So, it seems more pragmatic to work through the requirement with application code changes, rather than incurring the overhead of the dirty database at all times. We still have the early record retrieval (intra-session) requirement, but I'm hoping we can come up with a simpler way to deal with that. In addition, we need to fully understand any other requirements the dirty share implementation currently is meeting, and deal with those as well.

At this time, I don't have a solution in mind to replace the existing implementation, but I'm opening this issue to discuss ideas about how we can meet the same requirements with a simpler, more efficient solution.

#2 - 01/13/2023 02:49 AM - Greg Shah

How easy is it to separate the portions only used for the cross-session behavior (which is about the 4GL transaction isolation quirk) from the single-session behavior (used by the record nursery)? I'm hoping there is some win we can achieve to separate these. It seems like some amount of the processing will be for synchronization which is not needed.

At a minimum, for all cases except for the _____ table in that one application, we could use private databases and LOCK_MODE=0, right?

#3 - 01/13/2023 03:27 AM - Eric Faulhaber

Greg Shah wrote:

> How easy is it to separate the portions only used for the cross-session behavior (which is about the 4GL transaction isolation quirk) from the single-session behavior (used by the record nursery)? I'm hoping there is some win we can achieve to separate these. It seems like some amount of the processing will be for synchronization which is not needed.

We need someone to do a deep dive into the implementation to identify all the dependencies to answer that question. As I noted earlier, I have a recollection that there are other dependencies on the dirty database.

> At a minimum, for all cases except for the _____ table in that one application, we could use private databases and LOCK_MODE=0, right?

For the single-session early find behavior, yes, we could use private databases, but the change is more than a simple H2 configuration change. The code was written with a single, shared database in mind. Off the top of my head, at minimum the lazy table creation code will need to change, probably other areas as well. There's also a memory concern in duplicating all those databases, as well as a concern as to whether setting up and tearing down a database and the tables as sessions come and go will cancel out any benefit from less synchronization. We would need a way to test this to make sure we get a net benefit.

I would think using LOCK_MODE=0 would be a positive change, but we only get that if we first move to private databases.

#4 - 01/13/2023 10:06 AM - Greg Shah

- Assignee set to Stanislav Lomany

Please create a NopDirtyShareContext by extending DirtyShareContext and write each method as an empty implementation. Where needed return some safe value. We will use this to test the cost of dirty database on the current customer performance scenario. I'd like to get a read on this as quickly as possible.

#5 - 01/13/2023 10:07 AM - Greg Shah

Optimally this should be configurable in the directory so that we can turn it on or off without code changes.

#6 - 01/13/2023 10:18 AM - Greg Shah

FYI, the following are the existing (known) use cases for the dirty share database.

1. FIND query cross-session modification of query results to mimic the lack of transaction isolation in OE
2. FOR query cross-session notification of index changes to mimic the lack of transaction isolation in OE (uses the global event queue)
3. FIND for non-temp-table records in the record nursery

The first two cases are the _____ table case from that one application. We are trying to get rid of those cases for sure.

The third case is possibly still needed, but I think it is actually a mis-match with how we are handling flushing and validation. This is the case where a record is created in one buffer and all of the unique index components are not yet assigned/made unique. Then in another buffer, a FIND is done and in OE the record is found but in FWD it must come from the record nursery. I wonder if in OE the record actually exists in the database. If a FIND traverses the record, I guess it must be in an index somewhere. I don't think it can be in an index without being in the database. It suggests that in OE the unique index is not actually guaranteed on all records until some commit point where the validation has occurred. This may inform our approach to solving this problem, which in a perfect world would not involve having this dirty database.

#7 - 01/13/2023 11:43 AM - Eric Faulhaber

Greg Shah wrote:

> Please create a NopDirtyShareContext by extending DirtyShareContext and write each method as an empty implementation. Where needed return some safe value. We will use this to test the cost of dirty database on the current customer performance scenario. I'd like to get a read on this as quickly as possible.

Stas, please base these changes off the latest revision of branch 6129b.

#8 - 01/13/2023 02:15 PM - Stanislav Lomany

- Status changed from New to WIP

#9 - 01/13/2023 03:44 PM - Stanislav Lomany

> Please create a NopDirtyShareContext by extending DirtyShareContext and write each method as an empty implementation.

I created NopDirtyShareContext, but now I'm thinking: we can just always return null from DirtyShareFactory.getContextInstance in the dirty context is disabled. Or you need NopDirtyShareContext for profiling or something?

#10 - 01/13/2023 04:51 PM - Stanislav Lomany

- File nop-dirty.diff added

- File NopDirtyShareContext.java added

I'm done for today, but for the case you're in a hurry, I've attached option 1: NopDirtyShareContext as well as option 2: a diff making DirtyShareFactory.getContextInstance return null when necessary. If you want to use option 1 which is NopDirtyShareContext, you'll have to modify the diff file a bit.

#11 - 01/14/2023 07:17 AM - Greg Shah

Eric expected that the persistence layer will break badly if there is not a safe mocking version of the NopDirtyShareContext. Did you test this with any application? Hotel or one of the big customer apps?

#12 - 01/14/2023 04:59 PM - Greg Shah

Let's make sure that Hotel works properly with your changes.

#13 - 01/14/2023 05:00 PM - Greg Shah

The last time we measured it, I recall that about 10% of time was spent inside the dirty database. Removing it might be a very big deal.

#14 - 01/15/2023 06:38 AM - Stanislav Lomany

I've tested it in Hotel (I had to disable ConnectionManager.authenticate to run it) and found no persistence-related issues except a minor one on context cleanup, which I'll fix.

Greg, should I go with "disable dirty share" way rather than "NopDirtyShareContext" way?

#15 - 01/15/2023 03:19 PM - Stanislav Lomany

- File 7037.diff added

Please review the diff which allows to disable dirty sharing. I've tested trunk with this change on the big customers' apps and they seem to be working fine.

Set

```
<node class="container" name="persistence">
   <node class="boolean" name="dirty_share">
      <node-attribute name="value" value="FALSE"/>
   </node>
</node>
```

to disable dirty sharing.

#16 - 01/15/2023 04:36 PM - Greg Shah

Eric: Do you have any objections?

This seems safe to commit this in both trunk and in 6129b.

#17 - 01/15/2023 05:13 PM - Eric Faulhaber

Greg Shah wrote:

> Eric: Do you have any objections?
>
> This seems safe to commit this in both trunk and in 6129b.

No objection.

#18 - 01/16/2023 01:07 AM - Stanislav Lomany

> This seems safe to commit this in both trunk and in 6129b.

Committed as 6129b rev 14362...

#19 - 01/16/2023 01:59 AM - Constantin Asofiei

Eric, doesn't this disable the FIND for non-temp-table records in the record nursery case Greg mentioned? I ask because I recall encountering this in large customer app.

#20 - 01/16/2023 08:00 AM - Greg Shah

Constantin Asofiei wrote:

> Eric, doesn't this disable the FIND for non-temp-table records in the record nursery case Greg mentioned? I ask because I recall encountering this in large customer app.

Yes, it will disable the record nursery. This setting will not be safe (at least right now) for all apps. We also know that the ChUI regression app will fail without the cross-session portions.

For now we are trying to see:

- Can this be used for the current testcase?
- If so, what is the performance benefit?
- If there are problems, can we resolve them to obtain the performance benefit?

#21 - 01/16/2023 09:41 AM - Stanislav Lomany

Created task branch 7037a from FWD trunk revision 14477.

#22 - 01/16/2023 12:42 PM - Stanislav Lomany

7037a has been merged into the trunk as bzr revision 14478.

#24 - 01/19/2023 10:48 AM - Greg Shah

We have multiple applications which are dependent upon the record nursery (item 3 in #7037-6) but which do not (as far as we know) depend upon the cross-session support (items 1 an d2 in #7037-6).

The next step in this task is to cleanly disable all cross-session processing without affecting the record nursery.

#25 - 01/26/2023 02:09 PM - Stanislav Lomany

As far as I understand, the following should happen when dirty sharing is disabled in the directory:

1. Dirty database remains in place.
2. Dirty share manager remains in place.
3. Global event manager is not created and all interactions with it are disabled: from the dirty share manager as well as from AdaptiveComponent.
4. The following are the places where the dirty context is used. We have to disable this interactions or leave intact. I think we should do the following:

- AdaptiveFind.maybeInvalidate - disable
- PreselectQuery.coreFetch - disable
- RAQ.executeImpl - keep processDirtyResults processing and initialization of RAQ.dmoSorter, disable other interactions
- TxWrapper.end - keep

What do you think?

#26 - 01/30/2023 11:10 AM - Stanislav Lomany

One more thing: what cases that should be tested to make sure then remain working after cross-session dirty sharing has been disabled? Anything else except FIND by recid for non-temp-table records?

#27 - 01/30/2023 12:58 PM - Eric Faulhaber

Stanislav Lomany wrote:

> As far as I understand, the following should happen when dirty sharing is disabled in the directory:
>
> 1. Dirty database remains in place.
> 2. Dirty share manager remains in place.
> 3. Global event manager is not created and all interactions with it are disabled: from the dirty share manager as well as from AdaptiveComponent.
> 4. The following are the places where the dirty context is used. We have to disable this interactions or leave intact. I think we should do the following:
>
> - AdaptiveFind.maybeInvalidate - disable
> - PreselectQuery.coreFetch - disable
> - RAQ.executeImpl - keep processDirtyResults processing and initialization of RAQ.dmoSorter, disable other interactions
> - TxWrapper.end - keep
>
> What do you think?

As I review the code, it looks like we already did something like this a while ago. See the persist/dirty/DirtyShareSupport class. But this leaves dirty share enabled by default. There is this comment in that class:

```
      // The dirty share feature is enabled globally by default (i.e., all tables, all databases, except
      // temp-tables). This is the safer thing to do functionally, but it is NOT good for performance.
      // So, we allow this default behavior to be disabled in the directory, once the DMOs that really
      // need it, if any, have been identified and marked with the "dirty-read" annotation. When disabled
      // globally, only those DMOs will use the dirty-share features.
```

It may be that we want to invert this to disable it by default, and only enable it for those tables marked with the "dirty-read" annotation. Please track that annotation back and see where it comes from (probably a schema hint, IIRC).

Please review the current implementation as it is with the current feature persistence/disable-global-dirty-share set to true, and see if and where it differs from your summary of the behavior you would expect as you describe it above. From there, we can determine if we need to disable any processing that isn't covered by the current implementation.

#28 - 01/30/2023 01:01 PM - Eric Faulhaber

Stas, I'm sorry I did not remember this existing implementation. How does this overlap/conflict/differ with your recent changes?

#29 - 01/30/2023 03:03 PM - Stanislav Lomany

In DirtyShareSupport we effectively have forceDirtyShare always enabled, and, correct me if I'm wrong, no converted DMOs in the projects have "dirty-read" annotation. So dirty sharing is OFF for all tables, but DirtyShareSupport makes it ON. DirtyShareSupport affects DmoMeta.isDirtyRead: if isDirtyRead returns false (i.e. dirty sharing is OFF) then RecordBuffer.dirtyContext is null and dirty sharing is completely disabled (including sharing within the same session), which returns us to the starting point: cross-session sharing should be separated from sharing within the same session.

> How does this overlap/conflict/differ with your recent changes?

My current implementation is "disable everything" and DirtyShareSupport is "enable everything". I think we can have a single directory parameter with "disable" (default), "enable", and, if needed, "enable for all tables with dirty-read annotation" options.

#30 - 01/30/2023 05:43 PM - Eric Faulhaber

AFAICT, your assessment of how DirtyShareSupport works currently is correct.

It seems we need to support the following functionality:

- Cross-session use for the small set of tables (currently, only one I know of) which needs it. The existing use case emulates the atomicity of a sequence by taking advantage of this transaction isolation quirk.
  - Even though we could (maybe should) rewrite that application code to use a proper sequence, we should retain this capability in case we need it. So far, this has been rare, and I would hope it would remain so.
  - I don't think the global event notification is needed for this; the dirty share checking via FindQuery should be enough to support this case.
- Early record retrieval within the same session (the RecordNursery use case).
  - This is where a record is created (but not yet flushed) in one buffer, then retrieved in another buffer, within the same session. This is a distinct use case from the cross-session case.
  - This should also be uncommon, though not as rare as the cross-session case. We've seen a few instances of this.
  - We probably want to selectively enable this only for some tables, as we identify the cases.
  - Another alternative is to rework the original 4GL code to not use this data access pattern.
- Adaptive query/find invalidation, but within the current session only.
  - I think only the cross-session notification requires the dirty database and the global event queue, so we should be able to disable this in all cases.
    - This level of cross-session notification for adaptive find/query invalidation probably is overkill.
    - I don't think we've seen an application designed to actually rely on an uncommitted transaction in one application change FOR-EACH navigation in another. Someone please correct me if they remember a case.
  - I'm pretty sure everything intra-session goes through the ChangeBroker, via the RecordBuffer.reportChange method. That needs to survive (it is not really in scope w.r.t. the dirty share implementation, anyway).

So, the cross-session and intra-session use cases for dirty database must survive and are both still needed. Can we consider the intra-session use case to be a sub-set of the cross-session use case, or are they distinct / mutually exclusive? Either way, we need to distinguish which is needed. Today, dirty read is either on or off for a given table, but there is no distinction between these cases.

In terms of implementation, a shared database is needed for the cross-session case. A private database may be used for the intra-session case, but what is the overhead of this (i.e., constantly creating and dropping private databases and tables)? Is it similar to the legacy temp-table support (which seems to be faster using private databases)? And considering we will be doing it for a (hopefully) small subset of tables or for none at all, perhaps the overhead is negligible?

#31 - 01/31/2023 05:38 AM - Greg Shah

> Cross-session use for the small set of tables (currently, only one I know of) which needs it.

Tomasz may have found that there are multiple tables in that application which require the dirty database support. We can't put the list here in a public task.

Tomasz: Please post the final list in #6667 and link to it here.

Only the n_____ case was known and expected to need this transaction isolation quirk. I wonder if the other cases are relying upon a different aspect of dirty database.

> Early record retrieval within the same session (the RecordNursery use case).

I question the conclusions here. Do we have testcases that clearly demonstrate that the record is **not flushed but is returned back in index order**? As far as I understand it, OpenEdge cannot do that.

#32 - 01/31/2023 08:25 AM - Tomasz Domin

Greg Shah wrote:

> Cross-session use for the small set of tables (currently, only one I know of) which needs it.

> Tomasz may have found that there are multiple tables in that application which require the dirty database support. We can't put the list here in a public task.
>
> Tomasz: Please post the final list in #6667 and link to it here.

The updated list is here, but its not final yet: #6667-326
Once I have a final version I will post it here as well.

#33 - 02/02/2023 03:24 PM - Stanislav Lomany

FYI I've found uast/shared-dirty-records/p*.p and uast/shared-dirty-records/leaking*.p testcases which "demonstrate in- and inter-context record sharing in 4GL" and they do NOT work in FWD (they behave like dirty sharing is disabled).

#34 - 02/02/2023 03:26 PM - Tomasz Domin

Stanislav Lomany wrote:

> FYI I've found uast/shared-dirty-records/p*.p and uast/shared-dirty-records/leaking*.p testcases which "demonstrate in- and inter-context record sharing in 4GL" and they do NOT work in FWD (they behave like dirty sharing is disabled).

Can you please check #6948 ?
I made some naive changes to dirty-share cases to make it work with regression testing, not sure but it may help you.

#35 - 02/06/2023 06:45 PM - Stanislav Lomany

- File 7037b.diff added

Please review the diff (for the trunk 14478) which adds the following directory configuration parameters:

1. In addition to enabling/disabling cross-session dirty sharing, there's a parameter which allows enabling/disabling of dirty sharing within the same session.
2. Along with dirty-read table annotation, there's a dirty-intra-read annotation for in-session sharing.
3. In addition to the parameter which "forces" cross-session dirty sharing for all tables regardless of presence of dirty-read annotation, there's also a parameter which allows to "force" the new dirty-intra-read annotation.

Please note that the future dirty share changes (like #6948) should take into consideration DirtyShareSupport.enabled{Cross|Inter}Session parameter.

#36 - 02/10/2023 05:08 AM - Greg Shah

Ovidiu: Please review.

#37 - 02/13/2023 09:22 PM - Ovidiu Maxiniuc

Review of 7037b.diff.

A clean implementation. Nice job. The patch should work as described in note #7037-35.
Please add the history entries in each file header.

#38 - 02/15/2023 12:01 PM - Stanislav Lomany

Guys, I want to commit the dirty sharing changes, but I want to know what dirty share state should be for the existing projects? There're 5 new parameters which all default to false and therefore after the code commit, dirty sharing will be completely disabled. Is that the desired state?
The parameters:

1. Enable cross-session dirty sharing
2. Enable intra-session dirty sharing
3. "Force" cross-session dirty sharing: if cross-session dirty sharing is enabled, apply it to all tables rather only the tables annotated for cross-session dirty sharing.
4. "Force" intra-session dirty sharing: if intra-session dirty sharing is enabled, apply it to all tables rather only the tables annotated for intra-session dirty sharing.
5. Enable global notifications.

#39 - 05/14/2023 09:02 PM - Greg Shah

Please create a task branch and apply your changes. We are going to test this with various customer projects and with Hotel ChUI/GUI. If we need to have different config for specific projects then we will do that when this is merged to trunk.

#40 - 05/15/2023 08:59 AM - Stanislav Lomany

Created task branch 7037b from FWD trunk revision 14571.

#41 - 05/18/2023 10:33 AM - Stanislav Lomany

Rebased task branch 7037b from FWD trunk revision 14572.

#42 - 05/18/2023 10:37 AM - Greg Shah

Is this ready for review?

What is the %Done for this task?

#43 - 05/18/2023 10:41 AM - Stanislav Lomany

Yes, it is ready.

#44 - 05/18/2023 10:42 AM - Greg Shah

Does it default to the current approach? In other words, it won't cause changes in behavior for existing applications so it is safe to merge to trunk.

#45 - 05/18/2023 10:43 AM - Stanislav Lomany

Yes, it's safe.

#46 - 05/18/2023 10:43 AM - Greg Shah

Eric: Please review.

#47 - 05/18/2023 10:44 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Review

#48 - 05/18/2023 04:00 PM - Stanislav Lomany

Dirty sharing has been disabled in the directory for Hotel GUI (rev 264) and Hotel ChUI (rev 86).

#49 - 05/22/2023 05:49 PM - Roger Borrello

Stanislav Lomany wrote:

> Dirty sharing has been disabled in the directory for Hotel GUI (rev 264) and Hotel ChUI (rev 86).

Is the change to the directory the one noted in #7037-15? is that placed here?

```
    <node class="container" name="server">
      <node class="container" name="standard">
        <node class="container" name="persistence">
```

What aspects of the application need to be tested with 7037b to determine if this is working correctly for the application? Is it all or nothing?

#50 - 05/22/2023 06:09 PM - Eric Faulhaber

Code review 7037b, revs 14573-14574:

Everything looks good, but I have some questions, to make sure I understand some of the changes correctly:

In PreselectQuery and AdaptiveQuery, the flag DirtyShareSupport.enabledCrossSession is used to initialize to null a dirtyContext local variable which normally would hold a DirtyShareContext instance. The code previously was written to disable dirty share features if this local variable is null (which it would be, if dirty share was globally disabled AND the associated table did not force dirty sharing with the DMO's dirty-read annotation).

However, RandomAccessQuery.executeImpl does not use this idiom. Instead, the dirtyContext local variable is initialized using:

```
      DirtyShareContext dirtyContext = buffer.getDirtyContext();
```

...and later, when processing a LockTimeoutException, which might occur if another session deleted a record while we were trying to retrieve it, we have logic that looks like this:

```
                           if (dirtyContext != null &&
                               DirtyShareSupport.enabledCrossSession &&
                               dirtyContext.isDirtyDelete(entity, id, true))
                           {
                              ...
```

The if statement used to look like this:

```
                           if (dirtyContext != null && dirtyContext.isDirtyDelete(entity, id, true))
                           {
                              ...
```

Why are we not initializing dirtyContext to null in this method? Is it because we need to differentiate for the cross-session use in the lock timeout case, but not for the other cases in this method?

Is the dirtyIntraRead infrastructure in the Table annotation and in DmoMeta just there for future development? I don't see where any of it is used at runtime.

#51 - 05/22/2023 06:29 PM - Eric Faulhaber

Roger Borrello wrote:

> Stanislav Lomany wrote:
>
> > Dirty sharing has been disabled in the directory for Hotel GUI (rev 264) and Hotel ChUI (rev 86).
>
> Is the change to the directory the one noted in #7037-15?

No, it is the more granular aspects described in #7037-38.

> is that placed here?
> [...]

Yes, but I don't think the actual settings to be used in the directory are documented yet. Based on Stanislav's comment above, I expect they are in the directory configs in Hotel GUI rev 264 and Hotel ChUI rev 86. You can see the names the code reads from the directory in the static initializer at line 125 in the persist.dirty.DirtyShareSupport class in 7037b.

> What aspects of the application need to be tested with 7037b to determine if this is working correctly for the application? Is it all or nothing?

That's a great question, for which I don't have a simple answer. The places we've seen problems in the past are:

- concurrent (cross-session) access to a persistent table, but these are typically hard to capture, because they are timing sensitive;
- "early" queries for just-created records, intra-session but across buffers (i.e., a record is created in one buffer, is not yet fully initialized in a way that normally would cause it to be flushed to the database, while another buffer is used to find that infant record);
- others?

As to how we test specifically for these cases without a known recreate, I'm not sure. Tomasz has been finding numerous regressions in one application's regression tests which seem to be related to these dirty share features. We might find others in other automated regression test environments. Other than that, it's a matter of using of an application and knowing what to expect. In some cases, we may have abnormal ends with normal use.

Sorry I can't give you more exact instructions.

#52 - 05/23/2023 05:29 AM - Stanislav Lomany

Is the change to the directory ...

Roger, you can add these five parameters under persistence section:

```
<node class="container" name="persistence">
   <node class="boolean" name="dirty-cross-share">
     <node-attribute name="value" value="FALSE"/>
   </node>
   <node class="boolean" name="dirty-intra-share">
     <node-attribute name="value" value="FALSE"/>
   </node>
   <node class="boolean" name="force-dirty-cross-share">
     <node-attribute name="value" value="FALSE"/>
   </node>
   <node class="boolean" name="force-dirty-intra-share">
     <node-attribute name="value" value="FALSE"/>
   </node>
   <node class="boolean" name="dirty-share-global-notifications">
      <node-attribute name="value" value="FALSE"/>
   </node>
</node>
```

When all parameters are false then dirty sharing is completely disabled.

#53 - 05/23/2023 09:09 AM - Roger Borrello

Thanks, Stanislav. My application loaded up fine, but I don't think the latest revision of 7037b contains the 7277a merged to trunk at 14574, which is important for being able to use much of the application. But the initial startup of the application is very intense with loading up data objects, and that worked fine. Will you be rebasing this branch soon?

#54 - 05/23/2023 10:06 AM - Roger Borrello

Roger Borrello wrote:

> Thanks, Stanislav. My application loaded up fine, but I don't think the latest revision of 7037b contains the 7277a merged to trunk at 14574, which is important for being able to use much of the application. But the initial startup of the application is very intense with loading up data objects, and that worked fine. Will you be rebasing this branch soon?

I rebased my local branch, and everything went along without any issues. As far as I can tell, the settings you indicated for me to set in my directory are working just fine on this customer application.

#55 - 05/23/2023 12:06 PM - Stanislav Lomany

> Why are we not initializing dirtyContext to null in this method? Is it because we need to differentiate for the cross-session use in the lock timeout case, but not for the other cases in this method?

dirtyContext is not null if cross-session OR intra-session dirty-sharing is enabled. So in this case

```
if (dirtyContext != null &&
    DirtyShareSupport.enabledCrossSession &&
    dirtyContext.isDirtyDelete(entity, id, true))
    {
      ...
```

I assumed that dirtyContext.isDirtyDelete should be checked only in cross-session case. Hopefully it is correct.

> Is the dirtyIntraRead infrastructure in the Table annotation and in DmoMeta just there for future development? I don't see where any of it is used at runtime.

It is used in DmoMeta:

```
public boolean isDirtyRead()
{
   return DirtyShareSupport.enabledCrossSession && (DirtyShareSupport.forceCrossSession || dirtyRead) ||
          DirtyShareSupport.enabledIntraSession && (DirtyShareSupport.forceIntraSession ||
                                                                                ====>    dirtyIntraRead);
}
```

#56 - 05/23/2023 12:37 PM - Stanislav Lomany

> Will you be rebasing this branch soon?

Rebased task branch 7037b from FWD trunk revision 14584.

#57 - 05/31/2023 10:13 AM - Greg Shah

Eric: We are blocked on your code review.

#58 - 05/31/2023 01:48 PM - Eric Faulhaber

Greg Shah wrote:

> Eric: We are blocked on your code review.

How so? My code review was done in #7037-50. Everything looked good, I had some questions, Stanislav answered them. AFAIK, there haven been no additional changes, just rebasing.

#59 - 05/31/2023 01:51 PM - Stanislav Lomany

Should I rebase and commit?

#60 - 05/31/2023 02:08 PM - Eric Faulhaber

From my point of view, yes.

#61 - 05/31/2023 02:11 PM - Greg Shah

Go ahead and merge now.

#62 - 05/31/2023 02:43 PM - Stanislav Lomany

Rebased task branch 7037b from FWD trunk revision 14592.

#63 - 05/31/2023 03:20 PM - Stanislav Lomany

7037b has been merged into the trunk as bzr revision 14593.

#64 - 05/31/2023 03:24 PM - Greg Shah

- Status changed from Review to Test

#65 - 10/25/2023 10:40 AM - Greg Shah

- Related to Support #7963: disable dirty share database and test existing customer applications added

#66 - 10/25/2023 10:42 AM - Greg Shah

- Related to Feature #7964: investigate the benefits of disabling cross-session features of the dirty share database (without completely eliminating all dirty share functionality) added

Files

nop-dirty.diff	1.69 KB	01/13/2023	Stanislav Lomany
NopDirtyShareContext.java	14.1 KB	01/13/2023	Stanislav Lomany
7037.diff	2.41 KB	01/15/2023	Stanislav Lomany
7037b.diff	24 KB	02/06/2023	Stanislav Lomany