Database - Feature #8388

alternative implementation of same-session dirty sharing

03/04/2024 10:09 AM - Eric Faulhaber

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

Description

History

#1 - 03/04/2024 10:59 AM - Eric Faulhaber

The Problem:

In many cases, a record which is newly created in one buffer needs to be "seen" by another buffer performing a query (in the generic sense) in the same session, even though the new record may not yet be fully populated or even legal, in terms of the unique and not-null constraints defined for its table. We have the record in memory in a DMO, but that record has not yet been flushed to the database, because the requirements for validation and flushing have not yet been met.

The Current Solution:

The partial solution we have for this situation currently is to "flush" the records early via the record nursery. We get around the constraints as follows:

- for temp-tables, we make the constraints lenient (i.e., in the H2 temp table, legacy mandatory fields are not defined as not-null columns and legacy unique indices are not defined as unique), and we instead enforce these constraints in the FWD runtime;
- for persistent tables, we temporarily park the the potentially illegal DMO in the dirty database, because we can't make the constraints in the persistent database lenient, like we do for temp-tables (this is why "flush" is in quotes above -- the record doesn't really go to the persistent database).

There are problems with the current approach:

- The dirty share mechanism is limited and a performance problem:
  - It only works for converted FIND statements. The code to interleave "dirty" records with real results is complex already with single-table, single result queries; it was deemed too complex and potentially error-prone to implement this for single- and multi-table result sets, and performance of such an implementation was a concern.
  - It does not work if a query's where clause has an embedded CAN-FIND for another table, because the full state of that second table is not reflected in the dirty database.
  - It performs poorly, involving a dedicated, embedded database, lots of pessimistic locking, and many small H2 transactions.
- Temp-tables are not properly defined.
  - This is more a theoretical problem. As long as we enforce the constraints in the runtime correctly, application code should not care, and there aren't external dependencies on the temp-table schema.

Alternative Idea:

For temp-tables, we probably can live with the current approach.

For persistent tables, an alternative approach...

When it is deemed that early flushing is needed to satisfy the intra-session buffer visibility problem (how to do this is TBD):

- Define a temporary table (in the persistent database) with the same structure as the new record's persistent table, but without the not-null or uniqueness constraints.
  - Not sure whether there should be no indices at all, or the same indices, but with the unique ones neutered, as we do for temp-tables today.
- When the early flush is needed, actually flush the potentially unfinished record to this "shadow" temporary table. Since it has none of the constraints of the primary table, the flush will not raise any not-null or uniqueness errors.
- When a query is executed against this table, use a SQL union to fetch results from both the primary table and the temporary table. As long as the column structure is identical between the primary table and shadow temp-table, the result set will contain both existing and "infant" records. The database handles the sorting.
- When processing the results and caching the DMOs to the session cache, we need a way to differentiate between "real" records and infant records. Possibly use a negative primary key value for the infant records?
- When the new DMO is actually validated and flushed, remove the associated infant record from the shadow temp table.

TBD

- Performance:
  - There is overhead in creating, using, and dropping the shadow temp table.
  - The use of union complicates the query plan, which may result in poorly performing queries. These have to be overcome, if possible, with the correct SQL.
  - The use of this mechanism must be limited to cases where it is truly needed. We don't want to saddle the entire system with this kind of heavy CRUD overhead, when it should only be necessary in limited cases.
- Complexity:
  - The SQL for queries becomes more complex when injecting the use of union, especially for situations which already produce complex SQL (e.g., multi-table joins, sub-selects, etc.).
  - Different dialects will need different SQL (e.g., do we use CTEs or other SQL features?...)
- Other...

#2 - 03/12/2024 06:47 AM - Alexandru Lungu

I like the idea of keeping everything in the persistent table and do the query logic there. From my POV, these cases should be quite rare - only if the converted code operates on the same table with two different buffers, so we are forced to "leak" the visibility of a transient record to another record.

I think we should still think of how to leverage the different-session dirty sharing. Maybe we can layer them nicely, but certainly we need heavy testing.

Some things that were not covered:

- **Triggers**: we need to decouple the "logical" flushing to the temporary table (from the persistent database) from the WRITE trigger call. When we do the logical flush into the temporary table, we should still avoid doing the WRITE trigger - as that will be too early.
- **Commit time**: at some point, we actually need to move the records from the temporary table to the master table, right? This is a "physical" flush.
  - What happens on rollback (to savepoint eventually): I suppose we simply let it be. Nothing special.
  - I suppose this is the moment when we are going to actually call the WRITE triggers.
- **Updates** and **Deletes**:
  - for updates, we can have them in the temporary table (from the persistent database), but we are forced to "prioritize them". **This doesn't have a solution yet**.
  - for deletes, I suppose a delete is releasing the buffer, so this is "aggressively" flushing the delete anyway. We just need to be careful to delete from the master table and temporary table.