Database - Feature #7943

improvements in lock management for diagnostics and release of orphaned locks

10/19/2023 03:53 PM - Greg Shah

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

Related issues:

Related to Database - Feature #4400: add /* UUID */ in where clause comment f...	**New**

History

#3 - 10/19/2023 04:44 PM - Greg Shah

- Related to Feature #4400: add / UUID */ in where clause comment for every 4GL query so that SQL logging can easily be mapped back to the specific 4GL query being processed added*

#4 - 10/19/2023 04:49 PM - Greg Shah

A customer with a ChUI application (see #5290) is on an older version of FWD and had issues with locks being held long past when they should exist.

Our understanding is that the problem is most likely an orphaned lock, i.e., the InMemoryLockManager holds a lock even after the session which acquired it is gone (either having ended/died on its own or as the result of killing it from the admin UI). If this is not the case, and a lock is being held by a session that is still alive, then it very likely is:

1. a legitimate lock that should not be released; or
2. a lock that is being held improperly due to a bug in FWD runtime logic outside the lock manager; or
3. a lock that is being held improperly due to a bug in application logic; or
4. a lock that is being held improperly due to the thread-unsafe access that was identified in the lock manager.

We intend to fix category 4. In FWD v4, we did find and we believe we have fixed issues in category 2.

Until the root cause of the orphaned lock issue is determined and fixed, with will build a set of dignotics to identify orphaned locks and and a workaround solution to release those orphaned locks held by the lock manager.

This feature would be exposed as an option (possibly a menu item or button) in the administrative UI. Activating it would trigger a scan within the InMemoryLockManager of all the locks currently held. We would identify the session(s) holding each lock, and check whether each such session is still alive, via a new API added to the SecurityManager. If a session cannot be identified as active, the lock would be released for that defunct session only, without affecting other sessions, which in all likelihood are operating normally. The release would be "natural", as if the lock were being released by the session holding it, and any waiting threads would be notified via the normal process, so that they could acquire the lock according to the current rules.

The result of the orphaned lock cleanup would be logged and would be reported back to the administrative UI. The scanning and reporting is the diagnostics part o fthe task.

This feature most likely would need to synchronize on the lockTable instance variable for longer than is needed for regular lock manager access. However, we think it can be implemented to complete pretty quickly, so that users won't notice a disruption.

This administrative feature is intended as relief, but we realize it is not an ultimate solution. However, we think it will leave the server in a safe state with the minimal disruption to users, and without requiring a server restart.

In order to enhance the solution, we want to allow a given lock to be associated with specific converted 4GL code so that we can determine how the orphaned locks came to exist. We do not want to implement creation and storage of stack traces at lock creation as that would kill performance. Instead, I propose that we implement #4400 (which we have to do for another customer anyway). A slight addition to #4400 will solve our case here as well. The idea is that during conversion, if a flag is set, we add a UUID to all queries, create statements and other databae access statements that obtain locks. The runtime APIs would have variants added which allow passing these UUIDs (but the UUIDs are optional). This means that any code in the runtime that wants to provide diagnostics info (e.g. logging, in a SQL query, in the lock definition...) can record this UUID and report it. Then it is a trivial effort to search for that UUID in the application code and it will appear only in one place. In the case of locks being acquired automatically when a record is updated (e.g. on a field setter) we will just use the UUID of the data access statement (the query, create...) for the lock. We don't want to add a UUID to every field assignment.

The idea here is to implement a solution for figuring out the root cause of any orphaned lock while also providing a short term workaround that is safe.