Database - Feature #5883

use UUIDs to associate locations in the converted source code with lock operations

12/08/2021 02:27 PM - Greg Shah

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

Description

Related issues:

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

History

#1 - 12/08/2021 02:29 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*

#2 - 12/08/2021 02:49 PM - Greg Shah

In #5290, a customer is has orphaned locks. They are on a very old FWD version and we don't know if the problems are still existing or not. If the problems still exist, then we need some way to diagnose the problem.

The idea here is to create a kind of "log" that stores the list of locking operations (in order of execution) with enough additional info to be useful in diagnosis.

Information to log:

- location in the converted 4GL code
- session/user
- timestamp
- lock operation

The last ones of these are easy to determine at runtime (and we already may know these things). The location in the code is different. In the past, for specific test cases we have used a stack trace to capture the exact point in the code to which the lock operation was associated. But stack traces are very expensive. They cannot be used in a production system.

Instead of a stack trace, we can generate a UUID at conversion for each unique location like CREATE, RELEASE, every query, open scope on a buffer, CREATE TEMP-TABLE, QUERY-PREPARE() and possibly other cases. For each case, we would create some way to pass the UUID in to the runtime. Where possible we want to chain a setUUID(<generated_uuid>) into an existing set of calls so that it is clear, clean and simple. In some cases we may need to modify signatures for methods instead of chaining. The key point here is that we will be able to correlate any lock operation with a specific location in the code. This approach is fast and mostly just costs memory.

The buffer open scope case is a kind of "catch all" that is used instead of exploding our UUIDs for every DMO setter call. It will get us back to the source file in which the scope was opened, which should be close enough. Again, we are trying to balance utility with clean code.

We don't keep a log today, we just track the current operation. So the log aspect of this is meant to store the full sequence of operations. This needs to be super-efficient, so I'm expecting a simple POJO container for the log entry that has a reference to the next log entry instance. The log itself should probably just be a list that is self linked and can be maintained very very quickly. I'm not suggesting using any of the collections.

There are many runtime changes (esepcially in the lock manager) that have to happen to make this work. We also will need to expose this log as data in our lock report in the admin UI.