Database - Feature #1879

push lock manager back into database

10/31/2012 12:41 PM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	320.00 hours
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #4369: implement stateless FWD server clustering	**New**

History

#1 - 10/31/2012 12:45 PM - Eric Faulhaber

Currently, the default lock manager resides in memory within the P2J application server. All code which needs to comply with locking must therefore route through the P2J server JVM. Ideally, lock management should be in the database, so it is shared across servers AND it is easy for non-P2J code to integrate.

Issues:

- Databases will likely vary in their lock management implementations in a way that is not compatible with Progress
- We may not have access to the locking features we need in each backend
- We may not have access to the locking features we need through Hibernate
- An independent implementation using a standard database table to manage locks may be slow and still does not force external code to use it

Still, we have to try...

#2 - 11/16/2016 01:21 PM - Greg Shah

- Target version deleted (23)

#3 - 10/24/2019 03:06 PM - Greg Shah

- Related to Feature #4369: implement stateless FWD server clustering added

#4 - 10/24/2019 03:06 PM - Greg Shah

Based on a discussion with Eric, he notes that the primary differences in our in-memory implementation versus the PostgreSQL-level exclusive and share locks are these 4GL behaviors:

- EXCLUSIVE locks can be downgraded to SHARE locks based on buffer scopes that are broader than the transaction scope.
- SHARE locks canbe obtained outside of a transaction scope.

Explicit Locking in PostgreSQL describes session-level advisory locks which seem to be a good match for this use case:

> "PostgreSQL provides a means for creating locks that have application-defined meanings. These are called advisory locks, because the system does not enforce their use — it is up to the application to use them correctly. Advisory locks can be useful for locking strategies that are an

> awkward fit for the MVCC model. For example, a common use of advisory locks is to emulate pessimistic locking strategies typical of so-called “flat file” data management systems. While a flag stored in a table could be used for the same purpose, advisory locks are faster, avoid table bloat, and are automatically cleaned up by the server at the end of the session."

In addition, both regular locks and advisory locks can be inspected via the pg_locks system view. This might be the way to implement the _lock metadata and/or the lock builtin funcs.

We will **try** to implement this facility in a cross-database way BUT if a particular database can't map to this facility then it is OK to have this lock manager implementation only available on some databases.