Database - Feature #4024

PostgreSQL shared memory JDBC driver

04/01/2019 06:22 PM - Eric Faulhaber

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

History

#1 - 04/01/2019 06:40 PM - Eric Faulhaber

This concept is a more outside-the-box corollary to the idea behind #4012. In a system where lots of round trips to the database for small, fast queries are common, it seems to make sense to reduce the overhead of those round trips as much as possible. Progress has the advantage that it can use shared memory between the 4GL client and the database to quickly access records. This has led to the common 4GL development idiom of looping queries (e.g., FOR EACH), often with nested FIND statements within the body of the loop. While fast through shared memory in Progress, these cause an impedance mismatch with the set-oriented operations typical for JDBC and modern SQL databases. These were designed to better handle larger result sets, typically using more complex query statements which encapsulate more of the algorithm to isolate all the data needed for a unit of work.

The idea here would be to adapt the existing JDBC driver to do something similar: use shared memory to communicate between the JDBC driver and the database. This would be a fairly intrusive change which would require development on the database server as well. This would bypass the current client/server protocol and access records more directly. The usefulness of this approach is built upon the presumption that there is significant time spent in traveling to and from the database. It is essential that this presumption is confirmed through measuring this time. If this presumption is incorrect, this idea would of course not be worth the effort.

A drawback of this approach would be that the changes would most likely not be of interest to the larger PostgreSQL community, and thus it would be surprising if they were accepted back into the mainstream code base. Thus, this solution likely would involve the technical debt of carrying these custom changes across version upgrades.

Before attempting to outline any design for such an implementation, the first step is to establish a way to measure the actual overhead. The outcome of implementing #4012 suggests there may be less overhead than initially thought. As a first step, we should determine whether there is any logging capability in the existing JDBC driver which might help in this determination. If not, this can be added, since the driver is open source.