Database - Feature #4016

replace portions of Hibernate which are non-performant

03/31/2019 11:35 PM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #4015: upgrade to a newer Hibernate release	**Rejected**

History

#1 - 03/31/2019 11:43 PM - Eric Faulhaber

We maintain a lot of information in FWD's persistence layer which is redundant with information maintained by Hibernate. As we determine where we are spending too much time in Hibernate doing work that could be done more efficiently in FWD, we need to consider replacing those portions of Hibernate with a more efficient implementation in FWD. In some cases, Hibernate may already provide hooks for plug-in implementations, in other cases, we may need to introduce those ourselves.

Potential areas:

- Session implementation
- Second level cache
- Dirty checking infrastructure (to some degree we already short-circuit much of this work, but this could be expanded further)

There may be other areas as well. Actual targets will be determined by gathering performance data. New areas may become clear after we upgrade to a newer version of Hibernate.

#2 - 03/31/2019 11:44 PM - Eric Faulhaber

- Related to Feature #4015: upgrade to a newer Hibernate release added

#3 - 07/15/2019 09:59 PM - Eric Faulhaber

Based on further analysis, including using the persistence instrumentation (#4056), we have decided to remove as much of Hibernate as possible (ideally all of it), and replace it with a bespoke ORM implementation that is less heavyweight and designed specifically for FWD's needs. Some considerations:

- We already track so much about what has changed in a session, we can be more efficient about updates than we are today using Hibernate's dirty checking, even though we already have exploited some hooks to make it more efficient.
- Continue to use an HQL-compatible query language, at least for the preliminary WHERE and ORDER BY clauses that are emitted into converted business logic.
  - We will need to convert this into dialect-specific SQL.
  - Can we leverage/extend the HQL preprocessor for this?
  - Can we use various SQL templates for the limited set of query types we need and drop the WHERE and ORDER BY clauses in? May be complicated for joins and subqueries.
- Implement query/result caching that is more directly designed for FWD's needs.

- Need to manage/pool connections.
    - Expect to continue to use c3p0.
    - Longer life for connections to better amortize JDBC resources (prepared statements)?
- The DMO implementation classes were introduced primarily to work with Hibernate; ideally we will get rid of them.
    - Replace them with a runtime implementation (DMO ASM approach?), defined by the interface.
    - Can we get more performance from a more "raw" implementation? Idea is to transition data to/from BDT wrapper instances only when necessary (interfacing with business logic, internal work which requires 4GL semantics), but not for most internal use or for interfacing with the database. Have to look at all the transition points used today and see if we can reduce them.
    - Need to consider backward compatibility for existing installations.
- Drop separate XML for actual ORM mapping, leverage annotations within DMO interface. Also move existing legacy annotations to DMO interface from implementation class.
    - We do some custom manipulation of XML DOM in ORMHandler class today (e.g., to add mapping of computed columns for dialects which don't support upper and rtrim functions directly in indices); need to look at exactly what is being done here and see where this implementation should go.
- Need to implement record inflation from JDBC result set and flattening to prepared statement for interface with database. This is done with Hibernate custom type implementations today.
- Session factory concept goes away. We need a configuration that is easily mutable and naturally supports temporary tables in a more performant way than MutableSessionFactory.

#4 - 02/01/2021 09:18 AM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from New to Closed

Hibernate was removed as of trunk rev 11348.