Database - Support #4928

rewrite FieldReference to use lambdas instead of reflection

09/29/2020 09:49 AM - Greg Shah

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

Description

Related issues:

Related to Conversion Tools - Feature #1735: leverage lambda expressions (Jav...	**WIP**
Related to Base Language - Feature #6819: refactor FWD proxy implementation t...	**New**

History

#1 - 09/29/2020 09:49 AM - Greg Shah

- Related to Feature #1735: leverage lambda expressions (Java 8 "closures") to reduce code bloat added

#2 - 09/29/2020 10:57 AM - Greg Shah

At runtime we generate implementation classes to back the DMO interfaces. All of these classes have existing methods for all possible getters and setters, depending on the field types and extents. We used these today via reflection. In particular, at construction we lookup the Method instances:

```
      // get getter/setter either for an extent component or extent field
      this.extent = getExtent(dmoIface, property);
      bulk = bulk && (extent != null) && (index == null);
      this.getter = getGetter(dmoIface, property, bulk);
      this.isIdAccessor = ID.equals(property);
      this.setter = getSetter(dmoIface, property, bulk);
```

Although this resolution has some cost (lots of map lookups etc...), I think the reflection processing to build these maps is done previously and is a kind of cache. Still, if this is the only usage, then this can be dropped.

The bigger win is to eliminate the invocation time cost of using the Method by using lambdas instead.

For this to work, we would need to define functional interfaces for each possible method type (simple getter, indexed getter, bulk extent getter, simple setter, indexed setter, bulk extent setter). If we don't re-implement the external usage of FieldReference to use generics, then these can be all based on BDT instead of specific types. We may need variants for the NumberType vs long subscripts. Each possible method type would have its own functional interface. This enables a corresponding lambda to be used as a replacement for each Method instance.

Changing the internal usage from Method to lambda is pretty straightforward. The only trick here is how to replace the Method lookup with a lookup of the lambdas. The core idea here is that we can generate additional lookup code inside the DMO implementation bytecode. We could do it without map lookups if we generated the bytecode equivalent to this (Resolvable can be used for the simple getter case):

```
public Resolvable lookupGetter(String fld)
{
   Resolvable getter = null;

   switch (fld)
   {
      case "recid":  // this would need to use the configured id text instead of being hard coded to recid
         getter = this::rowId;
         break;
      case "f1":
         getter = this::getF1;
         break;
      ...
```

```
    }

    return getter;
}
```

So: both the lookup and the invocation will be very fast compared to the current approach. Additional lookup methods would be there for indexed getters and so forth...

#3 - 11/16/2022 05:19 AM - Greg Shah

- Related to Feature #6819: refactor FWD proxy implementation to use ReflectASM instead of Java Method reflection added