Database - Feature #4323

parse full FQL statements and convert them to SQL

09/10/2019 03:39 PM - Eric Faulhaber

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

Description

History

#1 - 09/11/2019 02:59 PM - Eric Faulhaber

- Subject changed from parse full HQL statements and convert them to SQL to parse full FWDQL statements and convert them to SQL

As part of the persistence performance overhaul, we are removing Hibernate from FWD. However, we still need an object-oriented query language from which we can generate dialect-specific SQL query statements. To that end, we will continue to use the general syntax of HQL, though we are no longer bound to its constraints. That is, we can modify the syntax to better fit FWD's needs.

We should remove references to "HQL" from our user documentation, such that it is clear that Hibernate is no longer part of the solution. However, such doc updates are outside the scope of this particular task. Maybe we refer to it as "FWDQL" (the more obvious "OQL" and "OOQL" already seem to be taken) and give credit to HQL for its syntax roots, but I don't want people thinking this is HQL anymore.

We can start with the HQL grammar we already have written for the HQLPreprocessor. Currently, it only handles the WHERE clause portion of an HQL statement. Let's rename that to fwdql.g. It may be that the current HQLPreprocessor gets absorbed into this new parser, but I'm not sure about that yet. The immediate objective is to be able to parse FWDQL statements and generate SQL from them.

The overriding objectives of this implementation are *correctness* and *performance*, but it (at least initially) does not need to be as complete as HQL. We just need to handle the types of queries we perform with FWD, not the full range of associations Hibernate supports, most of which we never use. I also want to support a JOIN syntax (not necessarily as complete as SQL, but beyond the theta-style only join you can do with HQL, without defining an ORM association). What I mean by this is I'd like to be able to express things like:

```
from Foo f join Bar b on b.prop1 = f.propA ...
```

and

```
from Foo f outer join Bar b on b.prop1 = f.propA ...
```

...without having explicitly to declare an association between entities Foo and Bar. The join keyword on its own will mean LEFT INNER JOIN in SQL terms; outer join will mean LEFT OUTER JOIN in SQL terms. **This join syntax does not need to be there on day one.** I just wanted to document this requirement while it's on my mind; it has long been missing from our tool box.

The translation from FWDQL to SQL needs to be *fast*. That means no use of TRPL, since introducing that runtime dependency will slow things down. We should figure out the optimal strategy to cache and re-use as much computed information as we can between query occurrences, so we can minimize the need to re-parse and re-translate the same FWDQL statements over and over. Maybe there is something conceptually we can use from the RuntimeJastInterpreter caching as a starting point? I'm not sure.

We can borrow functionality from our ORM implementation, where each DMO has its own SQL "template" for a select clause, which uses positional rather than name-based gets to read data from a result set.

I'm sure we will have more to add, but this should be enough to get started.

#2 - 09/11/2019 04:22 PM - Ovidiu Maxiniuc

A few questions before starting work:

- we do not have a FWDQL generator. Should I use the current conversion (HQL) instead?
- should the new solution work with Hibernate library (jar) removed from project?
- as you said, HQLPreprocessor already parses and analyses the HQL (FWDQL) string. I wonder if it wouldn't it be better to use it to do the actual conversion and get a SQL string as output? I do not remember post-processing the resulting string.

#3 - 09/11/2019 04:57 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> A few questions before starting work:
>
> - we do not have a FWDQL generator. Should I use the current conversion (HQL) instead?

Yes, any changes to static conversion (e.g., to leverage new features like the outer join) are out of scope for this task, unless you find something that could be changed in the conversion which would result in a better overall outcome and/or would make this task significantly easier to implement. If so, let's discuss specific ideas.

> - should the new solution work with Hibernate library (jar) removed from project?

Yes.

> - as you said, HQLPreprocessor already parses and analyses the HQL (FWDQL) string. I wonder if it wouldn't it be better to use it to do the actual conversion and get a SQL string as output? I do not remember post-processing the resulting string.

If it is easier to re-purpose this class than to write something from scratch, let's do it.

Also, let's use standard SQL as much as possible, but where we have to use dialect-specific syntax, target PostgreSQL (9.5 and higher) as the initial SQL dialect. I don't want to focus too much attention on architecting for different dialects initially. If it is easy to implement dialect-specific support as you go, go ahead with it. Otherwise, put TODOs in the code where different dialects would impact the resulting SQL. But the existing dialect hierarchy will need to change (it is based on Hibernate classes), and I don't want to get too involved with this until we have the basics working.

#4 - 09/12/2019 04:26 PM - Ovidiu Maxiniuc

I have investigated how much Hibernate is blended in FWD code: it's a lot: maybe 100 classes. Because of this, I think we should keep for the moment (until at least the basic statements are processed) the Hibernate library and only change the final part of the usecases: translating HQL to SQL and executing SQL. Of course, the translation will need additional information that is not available to the new FWD-ORM, so the point where they are setup in Hibernate will be duplicated for FWDQL.

Here are some things to be done (although this might not the right location for these notes):

- custom user types: the current implementations of UserType will not be useful and instead some custom-made class(es) will be added. Maybe their functionalities will be handled directly by BDT classes;
- dialects: we have the P2JDialect interface. Most likely it will become an abstract class;
- LE: Hibernate multi-level caching;
- LE: beside queries (SELECT stmts), the other statement types must also be translated.

#5 - 09/13/2019 07:35 PM - Ovidiu Maxiniuc

An intermediary status. I added support for gathering data from tables/fields annotations in a cached data structure and parsed the hql in Persistence.getQuery(), before passing the hqls to Hibernate's Context.getQuery(). I attempted to emit some queries and the result seem encouraging.

Here are two examples:

- ABL statement:

```
find first t100-ABC
```

- HQL (after being preprocessed by HQLPreprocessor):

```
from T100Abc_1_1Impl as t100Abc where t100Abc._multiplex = ?0 order by t100Abc.__it100F1 asc, t100Abc._mul
tiplex asc, t100Abc.id asc
```

- SQL generated by FWD:

```
SELECT
    t100Abc.t100_f4, t100Abc.t100_f3, t100Abc.t100_f2, t100Abc.t100_f1
FROM
    tt1 t100Abc
WHERE
    t100Abc._multiplex = ?
ORDER BY
    t100Abc.__it100F1 ASC, t100Abc._multiplex ASC, t100Abc.id ASC
```

- SQL generated by Hibernate:

```
select
    t100abc_1_0_.id as id0_, t100abc_1_0_._multiplex as column2_0_, t100abc_1_0_._originRowid as
 column3_0_, t100abc_1_0_._peerRowid as column4_0_, t100abc_1_0_._rowState as
 column5_0_, t100abc_1_0_.t100_f1 as t6_0_, t100abc_1_0_.__it100_f1 as column7_0_, t100abc_1_0_.t100_f2 as
 t8_0_, t100abc_1_0_.t100_f3 as t9_0_
from
    tt1 t100abc_1_0_
where
    t100abc_1_0_._multiplex=?
order by
    t100abc_1_0_.__it100_f1 asc, t100abc_1_0_._multiplex asc, t100abc_1_0_.id asc
limit ?
```

And the second example, a bit more complex:

- ABL statement:

```
for each Foo where prop-B < 25,
    each Bar where prop-2 > 0 and Foo.prop-A eq Bar.Prop-1
    break by Foo.prop-B:
```

- HQL (after being preprocessed by HQLPreprocessor):

```
select foo, bar from Foo_1_1Impl as foo, Bar_1_1Impl as bar where ((foo._multiplex = ?0) and (foo.propB <
25)) and ((bar._multiplex = ?1) and (bar.prop2 > 0 and (foo.propA = bar.prop1 or foo.propA is null and bar
.prop1 is null))) order by foo.id asc, foo._multiplex asc, bar.id asc
```

- SQL generated by FWD:

```
SELECT
    bar.prop_2, bar.prop_1, foo.prop_b, foo.prop_a
FROM
    tt2 foo
CROSS JOIN
    tt3 bar
WHERE
    ((foo._multiplex = ?) and (foo.prop_b < 25)) and ((bar._multiplex = ?) and (bar.prop_2 > 0 and
 (foo.prop_a = bar.prop_1 or foo.prop_a is null and bar.prop_1 is null)))
ORDER BY
    foo.id ASC, foo._multiplex ASC, bar.id ASC
```

- SQL generated by Hibernate:

```
select foo_1_1imp0_.id as id3_0_, bar_1_1imp1_.id as id2_1_, foo_1_1imp0_._multiplex as
 column2_3_0_, foo_1_1imp0_._originRowid as column3_3_0_, foo_1_1imp0_._peerRowid as
 column4_3_0_, foo_1_1imp0_._rowState as column5_3_0_, foo_1_1imp0_.prop_a as
 prop6_3_0_, foo_1_1imp0_.prop_b as prop7_3_0_, bar_1_1imp1_._multiplex as
 column2_2_1_, bar_1_1imp1_._originRowid as column3_2_1_, bar_1_1imp1_._peerRowid as
 column4_2_1_, bar_1_1imp1_._rowState as column5_2_1_, bar_1_1imp1_.prop_1 as
 prop6_2_1_, bar_1_1imp1_.prop_2 as prop7_2_1_
from
    tt2 foo_1_1imp0_
cross join
    tt3 bar_1_1imp1_
where
    foo_1_1imp0_._multiplex=? and foo_1_1imp0_.prop_b<25 and bar_1_1imp1_._multiplex=? and
 bar_1_1imp1_.prop_2>0 and (foo_1_1imp0_.prop_a=bar_1_1imp1_.prop_1 or (foo_1_1imp0_.prop_a is null) and
 (bar_1_1imp1_.prop_1 is null))
order by
    foo_1_1imp0_.id asc, foo_1_1imp0_._multiplex asc, bar_1_1imp1_.id asc
```

#6 - 09/16/2019 01:10 PM - Ovidiu Maxiniuc

I was working with join ing tables. I have the following issue: there is no on keyword in HQL. In fact, it is not really needed, the join expression is automatically extracted form the hibernate table relations. At this moment, our annotations do not have such information. Anyway, we can use the natural joins extracted at conversion time, but I really do not think this is enough for all possible queries, when the queries use theta style for custom user-defined predicates.

So I am thinking of adding this keyword ON as you used it in the 2nd example of the 1st note. Do you think it is useful in this form? I mean, will be FWD able to generate such FWDQL from compound queries?

The SQL syntax for OUTER joins requires the ON predicate. Although I believe that adding a ON true as the join predicate and keep the real one in the WHERE clause might do the workaround, I don't think this is a viable solution.

#7 - 09/16/2019 02:01 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I was working with join ing tables. I have the following issue: there is no on keyword in HQL. In fact, it is not really needed, the join expression is automatically extracted form the hibernate table relations. At this moment, our annotations do not have such information. Anyway, we can use the natural joins extracted at conversion time, but I really do not think this is enough for all possible queries, when the queries use theta style for custom user-defined predicates.

I want to be able to use the ON keyword without defining any associations between tables up front (including natural joins). Very similar to the use of JOIN ... ON ... in SQL.

> So I am thinking of adding this keyword ON as you used it in the 2nd example of the 1st note. Do you think it is useful in this form?

Yes, we can't to a server-side OUTER JOIN today because this is missing.

> I mean, will be FWD able to generate such FWDQL from compound queries?

Yes, optimized compound queries and multi-table preselect queries.

> The SQL syntax for OUTER joins requires the ON predicate. Although I believe that adding a ON true as the join predicate and keep the real one in the WHERE clause might do the workaround, I don't think this is a viable solution.

No, I'm talking about true [OUTER] JOIN ... ON ... syntax. This is a hole we have today, in that we cannot convert OUTER JOINs from 4GL to PreselectQuery (i.e., server-side join), they automatically get converted to CompoundQuery ("client"-side join).

#8 - 09/16/2019 02:44 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

Eric Faulhaber wrote:

> No, I'm talking about true [OUTER] JOIN ... ON ... syntax. This is a hole we have today, in that we cannot convert OUTER JOINs from 4GL to PreselectQuery (i.e., server-side join), they automatically get converted to CompoundQuery ("client"-side join).

This is the problem. The HQL does not have a JOIN ... ON .... This will be an FWDQL extension to HQL. If the conversion and then query optimizers will be able to generate these kind of constructs, the FWDQL-to-SQL utility will translate it.

#9 - 09/16/2019 03:12 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Eric Faulhaber wrote:
>
> > No, I'm talking about true [OUTER] JOIN ... ON ... syntax. This is a hole we have today, in that we cannot convert OUTER JOINs from 4GL to PreselectQuery (i.e., server-side join), they automatically get converted to CompoundQuery ("client"-side join).
>
> This is the problem. The HQL does not have a JOIN ... ON .... This will be an FWDQL extension to HQL.

Yes, that is exactly my intention. The lack of this syntax until now has limited us.

> If the conversion and then query optimizers will be able to generate these kind of constructs, the FWDQL-to-SQL utility will translate it.

That is the intention, but not right away. It will require changes in both areas, which are not the immediate priority. That is why I mentioned in #4323-1 that the addition of this syntax is not needed right away. Supporting the existing HQL syntax we currently use is higher priority.

#10 - 09/17/2019 05:18 PM - Ovidiu Maxiniuc

Two questions I thought about but I do not have the answer yet:

- how to handle EXTENTS (in denormalized case things seem simple, practically nothing to do);
- how do we handle computed columns? At this moment I do not have any clue about them, there is no annotation I can read, and the language converter is really not aware of their presence, so these columns will not be requested.

#11 - 09/19/2019 06:13 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Two questions I thought about but I do not have the answer yet:
>
> - how to handle EXTENTS (in denormalized case things seem simple, practically nothing to do);

I haven't committed in a while, because the build is broken in my working tree, as I've pulled the Hibernate imports out of most classes and I'm working through the dependencies. The table names are in an array named tables in the RecordMeta class. The first name is the "primary" table, and if there are any secondary tables for extent fields, they are in subsequent elements in the array, in ascending order of extent size (e.g., { foo, foo__5, foo__10 }). This is the current implementation, but it is not set in stone.

How you access this metadata is another matter, because each DMO implementation class will hold a RecordMeta instance specific to its layout, and these are not being generated yet. I have a FooImpl class mocked up for my early testing, but it is a few days out of date with my current changes, and it may be oversimplified for your needs. Also, the infrastructure classes which access this data are not finalized yet. I think committing these will pull in a lot of my other changes.

> - how do we handle computed columns? At this moment I do not have any clue about them, there is no annotation I can read, and the language converter is really not aware of their presence, so these columns will not be requested.

We were managing this by altering the Hibernate mapping documents on the fly for database dialects which need computed columns. I don't have a replacement idea for this yet. It is not needed for our immediate target, which is PostgreSQL, so please put TODO markers into your code where we will need to use computed columns for other dialects.

#12 - 09/19/2019 06:26 PM - Eric Faulhaber

In a recent email, you noted you had added support for field/column aliases. I'm not sure whether you were referring to parsing the FWDQL or generating the SQL, or both.

Please note that we want to access result set data *positionally*, rather than by name/alias, wherever possible, for performance. There may be cases where it is not practical/possible, because of the structure of a particular query, but to the extent it is, let's implement the SQL generation with this in mind.

See the class javadoc for the persist.orm.Loader class for a description of the data layout in the DMO implementation classes (actually, in the base class from which they inherit, BaseRecord).

I already generate common SQL for individual record inserts and reads, based on this layout. If this work can be extended to apply to queries as well, that would be great. I imagine some rework will be needed, at minimum because currently I use no table aliases in the statements I generate, whereas this will be necessary for many queries, especially joins.

#13 - 10/04/2019 01:50 PM - Eric Faulhaber

Ovidiu, I'm getting the following compile errors with 4011a/11342:

```
[ant:javac] /home/ecf/projects/p2j_branch/4011a/src/com/goldencode/expr/ExpressionLexer.java:171: error: canno
t find symbol
[ant:javac]          NumberFormat    lfmt       = NumberFormat.getInstance();
[ant:javac]          ^
[ant:javac]   symbol:   class NumberFormat
[ant:javac]   location: class ExpressionLexer
[ant:javac] /home/ecf/projects/p2j_branch/4011a/src/com/goldencode/expr/ExpressionLexer.java:171: error: canno
t find symbol
[ant:javac]          NumberFormat    lfmt       = NumberFormat.getInstance();
[ant:javac]                                       ^
[ant:javac]   symbol:   variable NumberFormat
[ant:javac]   location: class ExpressionLexer
[ant:javac] /home/ecf/projects/p2j_branch/4011a/src/com/goldencode/expr/ExpressionLexer.java:172: error: canno
t find symbol
[ant:javac]          NumberFormat    cfmt       = NumberFormat.getInstance();
[ant:javac]          ^
[ant:javac]   symbol:   class NumberFormat
[ant:javac]   location: class ExpressionLexer
[ant:javac] /home/ecf/projects/p2j_branch/4011a/src/com/goldencode/expr/ExpressionLexer.java:172: error: canno
t find symbol
[ant:javac]          NumberFormat    cfmt       = NumberFormat.getInstance();
[ant:javac]                                       ^
[ant:javac]   symbol:   variable NumberFormat
```

Are you seeing these? Is there a reason you removed the import statement for java.text.* from the ExpressionLexer header in expression.g? I can add it back, but I didn't want to conflict with any outstanding changes you may have in that file.

#14 - 10/04/2019 01:54 PM - Eric Faulhaber

- Subject changed from parse full FWDQL statements and convert them to SQL to parse full FQL statements and convert them to SQL

BTW, Greg has decreed that "FWDQL" should be "FQL" :)

Please work that name change back into the code, file names, doc, etc. I tend to agree that "FQL" is less likely to cause mumbling than "FWDQL".

#15 - 10/04/2019 01:57 PM - Ovidiu Maxiniuc

- Subject changed from parse full FQL statements and convert them to SQL to parse full FWDQL statements and convert them to SQL

Sorry, Eric. I cleaned the .g file only looking at the ExpressionParser. Indeed, I was too eager to drop the java.text.*. I added it back. Please see 11343.

#16 - 10/04/2019 01:59 PM - Ovidiu Maxiniuc

- Subject changed from parse full FWDQL statements and convert them to SQL to parse full FQL statements and convert them to SQL

I forgot to CTRL+F5 :(.

#17 - 10/04/2019 02:16 PM - Eric Faulhaber

Is there a new class DDLGeneratorWorker that was supposed to be checked in?

```
...
Caused by: com.goldencode.p2j.cfg.ConfigurationException: Unable to register worker: com.goldencode.p2j.persis
t.orm.DDLGeneratorWorker
    at com.goldencode.p2j.pattern.ConfigLoader.worker(ConfigLoader.java:763)
    at com.goldencode.p2j.pattern.ConfigLoader.processChildElements(ConfigLoader.java:714)
    at com.goldencode.p2j.pattern.ConfigLoader.ruleSet(ConfigLoader.java:1058)
    at com.goldencode.p2j.pattern.ConfigLoader.loadImpl(ConfigLoader.java:557)
    ... 13 more
...
```

#18 - 10/04/2019 04:01 PM - Ovidiu Maxiniuc

I committed latest changes to 4011a/rr11344.

If you have an old ddl directory, please backup it. The conversion will overwrite the old table creation files (.sql). Compare them to see the current state.

Note that, by default, the *Impl.java and *.hbm files will NOT be created. Use the following values in p2j.cfg.xml file to override them:

```
    <parameter name="load-condition"                  value="force-dmo-impl,force-dmo-hbm" />
    <parameter name="no-annotations-in-dmo-interfaces" value="true" />
<!-- default is FALSE, use TRUE to generate DMO interfaces WITHOUT the new annotations -->
```

#19 - 10/04/2019 05:35 PM - Eric Faulhaber

Are there any other project-level changes that need to be made, beside those in your last note? I'm trying to convert something in the testcases project, and it still seems to be using the Hibernate-driven DDL generation:

```
--------------------------------------------------------------------------------
Generate metadata DDL scripts for schema 'standard' and target DB 'h2'
--------------------------------------------------------------------------------

./data/namespace/standard.p2o
Generating INDEX DDL for table meta_db...
Generating INDEX DDL for table meta_field...
Generating INDEX DDL for table meta_file...
Generating INDEX DDL for table meta_index...
Generating INDEX DDL for table meta_index_field...
Generating INDEX DDL for table meta_lock...
Reading merge DMO definitions...
EXPRESSION EXECUTION ERROR:
---------------------------
exporter = create("org.hibernate.tool.hbm2ddl.SchemaExport", config)
           ^  { java.lang.reflect.InvocationTargetException }
---------------------------
Elapsed job time:  00:00:00.166
ERROR:
com.goldencode.p2j.pattern.TreeWalkException: ERROR!  Active Rule:
-----------------------
      RULE REPORT
-----------------------
Rule Type :   POST
Source AST:  [ _meta ] DATA_MODEL/ @0:0 {55834574849}
Copy AST  :  [ _meta ] DATA_MODEL/ @0:0 {55834574849}
Condition :  exporter = create("org.hibernate.tool.hbm2ddl.SchemaExport", config)
Loop      :  false
--- END RULE REPORT ---

    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1070)
    at com.goldencode.p2j.convert.TransformDriver.processTrees(TransformDriver.java:587)
    at com.goldencode.p2j.convert.ConversionDriver.generateDDL(ConversionDriver.java:694)
    at com.goldencode.p2j.convert.ConversionDriver.middle(ConversionDriver.java:507)
    at com.goldencode.p2j.convert.TransformDriver.executeJob(TransformDriver.java:868)
    at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:991)
Caused by: com.goldencode.expr.ExpressionException: Expression execution error @1:12
    at com.goldencode.expr.Expression.execute(Expression.java:484)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:497)
    at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:585)
    at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:98)
    at com.goldencode.p2j.pattern.PatternEngine.apply(PatternEngine.java:1652)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1531)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1479)
    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1034)
    ... 5 more
Caused by: java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
    at com.goldencode.p2j.pattern.CommonAstSupport$Library.create(CommonAstSupport.java:782)
    at com.goldencode.p2j.pattern.CommonAstSupport$Library.create(CommonAstSupport.java:705)
    at com.goldencode.expr.CE5346.execute(Unknown Source)
    at com.goldencode.expr.Expression.execute(Expression.java:391)
    ... 12 more
Caused by: org.hibernate.HibernateException: Could not instantiate given dialect class: com.goldencode.p2j.per
sist.dialect.P2JH2Dialect
    at org.hibernate.dialect.Dialect.instantiateDialect(Dialect.java:247)
    at org.hibernate.dialect.Dialect.getDialect(Dialect.java:233)
    at org.hibernate.tool.hbm2ddl.SchemaExport.<init>(SchemaExport.java:171)
    at org.hibernate.tool.hbm2ddl.SchemaExport.<init>(SchemaExport.java:156)
    ... 20 more
Caused by: java.lang.ClassCastException: com.goldencode.p2j.persist.dialect.P2JH2Dialect cannot be cast to org
.hibernate.dialect.Dialect
    at org.hibernate.dialect.Dialect.instantiateDialect(Dialect.java:241)
    ... 23 more
```

#20 - 10/07/2019 10:21 AM - Ovidiu Maxiniuc

These are generateDDL() calls for the metadata database. I am working on it right now.

#21 - 10/10/2019 02:50 PM - Ovidiu Maxiniuc

Eric,

I am trying to clean-up the code after DDL changes and I am not sure whether we need the merge-dmo-root setting any more in p2j.cfg.xml. It points to a set of resources that are not available.
The information that was obtained from loading hbms in Hibernate is the same we put in the annotations of DMO interfaces. After these are generated, the DDLs for creating tables, sequences and indexes are dumped to same locations (dialect specific, as configured in p2j.cfg.xml).

Does it make sense that a similar merging operation to be implemented with the new changes?

#22 - 10/10/2019 04:40 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I am trying to clean-up the code after DDL changes and I am not sure whether we need the merge-dmo-root setting any more in p2j.cfg.xml. It points to a set of resources that are not available.
> The information that was obtained from loading hbms in Hibernate is the same we put in the annotations of DMO interfaces. After these are generated, the DDLs for creating tables, sequences and indexes are dumped to same locations (dialect specific, as configured in p2j.cfg.xml).

I'm not that familiar with the details of the DMO merge feature. Constantin, this is about merging in hand-coded DMO changes with the conversion process, right? With the removal of Hibernate, we are no longer generating HBM files. All that information, along with the legacy information stored in DMO implementation class annotations, has been moved to the DMO interfaces. The dmo-index.xml file will survive for now, but I want to remove that in the not-too-distant future as well, since it is mostly redundant with the DMO interface annotations.

> Does it make sense that a similar merging operation to be implemented with the new changes?

Yes, but it is lower priority. Is merge-dmo-root only about the HBM files? We will still need some way to merge hand-written DMOs with conversion output. I don't know ATM what the best way is to do that. Let's come back to this after the core ORM functionality is working. Please disable the merge step so it does not block conversion, but output an informational message for that step so it reminds us to deal with this.

#23 - 10/22/2019 07:20 PM - Eric Faulhaber

Ovidiu, in your last commit to 4011a, you removed all the default methods from Record which were based on your earlier TempTableRecord interface, and you removed the @Override annotations from their overriding implementations in TempRecord.

This design was deliberate on my part, to avoid casting Record to TempRecord. Did you find a problem with it? If so, please let me know your rationale. Otherwise, I'd like to change it back.

#24 - 10/22/2019 07:59 PM - Eric Faulhaber

Eric Faulhaber wrote:

> ... I'd like to change it back.

FYI, I've done this in my local working copy. So in case you agree, please don't make the changes, as it will duplicate effort.

#25 - 10/23/2019 08:47 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu, in your last commit to 4011a, you removed all the default methods from Record which were based on your earlier TempTableRecord interface, and you removed the @Override annotations from their overriding implementations in TempRecord.

The _rowState, _originRowid, and the other are specific to PDS temp-tables. This makes it natural that they are TempRecord level.

> This design was deliberate on my part, to avoid casting Record to TempRecord. Did you find a problem with it? If so, please let me know your rationale. Otherwise, I'd like to change it back.

I know that that this was your intention, but I do not understand why is the casting a bad thing? We need to have some covariance here ie the TemporaryBuffer's current record is always a TempRecord. This way the case would not be necessary. I tried to make it so by overriding getCurrentRecord() to return a TempRecord. The problem is RecordBuffer, where some methods like copy(), which, in order to minimize the code duplication was "conditioned" to handle temp-tables, instead of overriding it in TemporaryBuffer. This is may fault, I introduced this with PDS support, but maybe this is the chance now to make it right. Maybe we should discuss this in more details.

#26 - 10/23/2019 12:40 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> The problem is RecordBuffer, where some methods like copy(), which, in order to minimize the code duplication was "conditioned" to handle temp-tables, instead of overriding it in TemporaryBuffer. This is may fault, I introduced this with PDS support, but maybe this is the chance now to make it right.

This and all the casting that was going on in some of the TemporaryBuffer methods with newly added dataset implications was mainly what drove me to try to consolidate the types into one. However, I do see your point; perhaps I went a step too far. I'll revert back to your latest changes.

BTW, your override of RecordBuffer.getCurrentRecord in TemporaryBuffer was interesting... I did not realize the compiler allowed an override of a method to return a different type. Hadn't come across that before, so thanks for teaching me something new!

#27 - 10/23/2019 04:17 PM - Eric Faulhaber

The one thing I really don't like, however, is the presence of casts to TemporaryBuffer within RecordBuffer methods. A parent class should not have references to a specific subclass, IMO.

#28 - 10/24/2019 10:19 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> The one thing I really don't like, however, is the presence of casts to TemporaryBuffer within RecordBuffer methods. A parent class should not have references to a specific subclass, IMO.

Exactly, this is what I expressed in note 25 by:

> The problem is RecordBuffer, where some methods like copy(), which, in order to minimize the code duplication was "conditioned" to handle temp-tables, instead of overriding it in TemporaryBuffer.