Database - Bug #2535

enhancing datatype conversion functions

03/10/2015 11:34 AM - Ovidiu Maxiniuc

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#1 - 03/10/2015 11:57 AM - Ovidiu Maxiniuc

I just discovered that we don't cover all the conversions between datatypes. We need to implement additional constructors for classes extending BDT to allow conversion from FieldReference objects. Here are more details.
Consider the following code (based on code from a customer application):

```
define temp-table tt1
    field f1-dte as date
    field f2-int as integer
    field f3-dte as date
    field f4-chr as char
    field f5-dec as decimal.

define temp-table tt2
    field f1-dte as date
    field f2-int as integer
    field f3-int as integer
    field f4-chr as character.

FOR
    EACH tt1
       WHERE  tt1.f1-dte = 02/05/05
         AND  tt1.f2-int = 10
         AND  tt1.f3-dte = 04/04/05
         AND  tt1.f4-chr = 'ABC' NO-LOCK,
    FIRST  tt2
       WHERE  tt2.f1-dte = 02/05/05
         AND  tt2.f2-int = 10
         AND  tt2.f4-chr = 'ABC'
         AND  (tt1.f5-dec = 0
            OR tt2.f3-int = INTEGER(tt1.f5-dec))
```

It will be converted to

```
query0 = new CompoundQuery(false, false);
query0.addComponent(new AdaptiveQuery(tt1, "tt1.f1Dte = '2005-02-05' and tt1.f2Int = 10 and tt1.f3Dte = '2005-04-04' and upper(tt1.f4Chr) = 'ABC'", null, "tt1.id asc", LockType.NONE));
query0.addComponent(new RandomAccessQuery(tt2, "tt2.f1Dte = '2005-02-05' and tt2.f2Int = 10 and upper(tt2.f4Chr) = 'ABC'", whereExpr0, "tt2.id asc"), QueryConstants.FIRST);
```

where

```
    WhereExpression whereExpr0 = new WhereExpression() {
       public Object[] getSubstitutions() {
          return new Object[] {
```

```
            new IntegerExpression() {
               public integer execute() {
                   return new integer(new FieldReference(tt1, "f5Dec"));
               }
            }
         };
      }
```

This is not compilable because we don't have such integer constructor at this moment. The issue was not visible by the compiler because this query is generated at runtime and only the classes that handle dynamic queries will encounter it.

The implementation is as simple as adding add a new constructor with at least a single Accessor parameter, because using FieldReference would add an unwanted reference to persistence. A second optional parameter of type character can be used to specify the format to be used during the conversion.
The affected datatypes are integer, int64, decimal, date, datetime, datetimetz and logical.
The STRING function is implemented as a static method valueOf in character class and it also must be overloaded.

#2 - 03/10/2015 12:15 PM - Eric Faulhaber

- Target version set to Milestone 11

#3 - 03/10/2015 01:35 PM - Ovidiu Maxiniuc

- File integer-function-1.p added

Eric,

I have an issue regarding the implementation of these methods. At first I implemented as simple as this:

```
public integer(Accessor fieldRef)
{
   assign(fieldRef.get(), false);
}
```

Taking into account that these methods/c'tors will only be called with a in-place newly constructed FieldReference, the argument will never be null. But the problem occurs when the fieldRef refers a buffer that is not yet loaded with a record from database at the time the loop is declared. The P2J will print No tt1 record is available in message are and the procedure will end (see the attached testcase).

This happens because the RandomAccessQuery c'tor will try to resolveArgs() that will invoke the resolveArgs() of WhereExpression, whereExpr0 that will getSubstitutions() and resolve each item of the array in order to keep them in currentArgs. At this moment, the new c'tor will attempt to access the field referenced by fieldRef and an noted above, the buffer is not yet available (or alternatively, a record from an old query may be referred).
This happens long before query0.iterate(); has the chance to actually load the correct record form database.

To prevent this, I tried to rewrite the constructor as follows:

```
public integer(Accessor fieldRef)
{
   BaseDataType bdt;
   boolean ignore = ErrorManager.isIgnore();
```

```
      ErrorManager.setIgnore(true);
      try
      {
         bdt = fieldRef.get();
      }
      finally
      {
         // restore ignore value
         ErrorManager.setIgnore(ignore);
      }
      assign(bdt, false);
   }
```

Ie. to wrap the fieldRef.get() in an error-ignored temporary bracketing so that the error 91, No tt1 record is available will not raised.

This will work just fine for my toy-procedure, but I am not sure that it is correct to ignore this error every time. In fact I feel that is totally wrong but at this moment I don't have other solution except for delaying the RandomAccessQuery to resolveArgs() until .. don't know when.

#4 - 03/10/2015 02:39 PM - Eric Faulhaber

- Assignee set to Ovidiu Maxiniuc

You are right: we don't want to mask this error always, because there may be times when it represents a legitimate error.

It seems the problem here is that this conversion is simply broken:

```
public integer execute() {
   return new integer(new FieldReference(tt1, "f5Dec"));
}
```

The whole point of using FieldReference in these situations is to defer evaluation until the WhereExpression base class decides it is safe to resolve the value, which in the case of FieldReference and other Resolvable parameters doesn't happen until WhereExpression.evaluate. By wrapping this in a call to an integer constructor in the generated getSubstitutions method, we have broken that deferral, as you've determined in your debugging.

Adding the proposed new BDT subclass c'tors won't fix this, unless they preserve the deferred nature of the Accessor passed into them. But we don't want to fundamentally change these data types to deal with this edge case.

I think the most correct solution is to refactor the conversion of the WhereExpression to look something like this:

```
WhereExpression whereExpr0 = new WhereExpression()
{
   public Object[] getSubstitutions()
   {
      return new Object[]
      {
         new FieldReference(tt1, "f5Dec");
      };
   }
   ...
}
```

...and have the integer c'tor (the existing one which accepts decimal) called in the converted evaluate method.

Please post what that generated evaluate method looks like currently, and let's figure out how it needs to look to support this semantic correctly.

#5 - 03/10/2015 02:53 PM - Ovidiu Maxiniuc

- Assignee deleted (Ovidiu Maxiniuc)

The generated evaluate method for WhereExpression whereExpr0 looks currently like this:

```
public logical evaluate(final BaseDataType[] args)
{
   return or(isEqual(tt1.getF5Dec(), 0), new LogicalExpression()
   {
      public logical execute()
      {
         return isEqual(tt2.getF3Int(), (integer) args[0]);
      }
   });
}
```

I guess we should keep the new constructor and replace the return line like:

```
return isEqual(tt2.getF3Int(), new integer(args[0]);
```

#6 - 03/10/2015 02:54 PM - Ovidiu Maxiniuc

- Assignee set to Ovidiu Maxiniuc

#7 - 03/10/2015 03:28 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I guess we should keep the new constructor and replace the return line like:
>
> ```
> return isEqual(tt2.getF3Int(), new integer(args[0]);
> ```

I think this proposed conversion is correct, but we don't need the new constructors. Each data type should already have a c'tor that accepts BaseDataType (please confirm this). Assuming we make the getSubstitutions conversion change above, args[0] should already be a decimal value by the time it is passed into the generated evaluate(BDT[]) method. It will have been obtained by resolving the FieldReference object returned by getSubstitutions (see WhereExpression.evaluate(), specifically line 197).

#8 - 03/10/2015 03:30 PM - Eric Faulhaber

- Project changed from Base Language to Database

#9 - 03/11/2015 10:45 AM - Ovidiu Maxiniuc

Yes, this is correct, the new constructors are not really needed.
I checked all datatypes classes and they all have a c'tor that accepts BaseDataType.

The fix for moving the field evaluation along with the function that use it as parameter is done by changing the evalLib("function_calls") or from line 196 of annotations/where_clause_post2.rules. I am working to rewrite this to something that can identify subtrees functions calls that don't contain references to fields that are not part of the current buffer. These subtrees must be kept in getSubstitutions().
If the subtree contains references to this kind of buffers, they are moved to evaluate() method and only the field references will get an entry in the getSubstitutions() returned array.

If the the tree would have already been processed some annotation could be in handy to identify these cases. But in walk-rules stage I must do an iteration of the subtree to detect non current-buffer field references.

#10 - 03/11/2015 02:56 PM - Ovidiu Maxiniuc

- File om_upd20150311a.zip added

With the attached update, queries like:

```
FOR
   EACH tt1
      WHERE  tt1.f1-dte = 02/05/05
        AND  tt1.f2-int = 10
        AND  tt1.f3-dte = 04/04/05
        AND  tt1.f4-chr = "0'005" NO-LOCK,
   FIRST  tt2
      WHERE  tt2.f1-dte = 02/05/05
        AND  tt2.f2-int = 10
        AND  tt2.f4-chr = "0'005"
        AND  (tt1.f5-dec = 0
           OR tt2.f4-chr = STRING(INTEGER(tt1.f5-dec), "9'999")
           OR tt2.f3-int = INTEGER(k)
           OR tt2.f1-dte + 1 = today):
```

gets converted to:

```
   WhereExpression whereExpr0 = new WhereExpression() {
      public Object[] getSubstitutions() {
         return new Object[] {
            new FieldReference(tt1, "f5Dec"),
            new IntegerExpression() {
               public integer execute() {
                  return new integer(k);
               }
            },
            new DateExpression() {
               public date execute() {
                  return date.today();
               }
            }
         };
      }

      public logical evaluate(final BaseDataType[] args) {
         return or(or(or(or(isEqual(tt1.getF5Dec(), 0), new LogicalExpression() {
```

```
            public logical execute() {
               return isEqual(tt2.getF4Chr(), valueOf(new integer((decimal) args[0]), "9'999"));
            }
         }), new LogicalExpression() {
            public logical execute() {
               return isEqual(tt2.getF3Int(), (integer) args[1]);
            }
         }), new LogicalExpression() {
            public logical execute() {
               return isEqual(tt2.getF3Int(), k);
            }
         }), new LogicalExpression() {
            public logical execute() {
               return isEqual(date.plusDays(tt2.getF1Dte(), 1), (date) args[2]);
            }
         });
      }
   };

query0 = new CompoundQuery(false, false);
query0.addComponent(new AdaptiveQuery(tt1, "tt1.f1Dte = '2005-02-05' and tt1.f2Int = 10 and tt1.f3Dte = '2005-
04-04' and upper(tt1.f4Chr) = '0'005'", null, "tt1.id asc", LockType.NONE));
query0.addComponent(new RandomAccessQuery(tt2, "tt2.f1Dte = '2005-02-05' and tt2.f2Int = 10 and upper(tt2.f4Ch
r) = '0'005'", whereExpr0, "tt2.id asc"), QueryConstants.FIRST);
```

I am not sure I wrote the function call annotation in the right place (index_selection.rules) but it can be moved.
Please review.

#11 - 03/13/2015 03:56 PM - Eric Faulhaber

Code review 0311a:

> I am not sure I wrote the function call annotation in the right place (index_selection.rules) but it can be moved.

Right, I don't think index_selection.rules is the most appropriate place for these changes, since they are about query substitution parameters, not index selection. This processing probably best belongs in where_clause_prep.rules, which is similar in structure to index_selection.rules, and which is where we do a lot of annotation regarding query substitution parameters.

I think the rest of the update is OK, but it's hard to tell whether it will cause broader changes than what you intend. This should become clearer with conversion regression testing.

#12 - 03/13/2015 05:33 PM - Ovidiu Maxiniuc

- File om_upd20150313b.zip added

Logic that was incorrectly placed in index_select.rules moved to where_clause_prep.rules.

#13 - 03/13/2015 05:48 PM - Eric Faulhaber

Code review 0313b:

Just thought of one edge case which might be problematic: consider a WHERE clause with a single, non-current-buffer field (would have to be a FIELD_LOGICAL) and nothing else. Wouldn't this mean the field's parent is the KW_WHERE node, which would be marked with the "where_not_subst" annotation? Perhaps a silly case which would better be refactored differently...but would it break the update?

Otherwise looks good. If it needs to be fixed to deal with the above case, please do so. Then, please conversion regression test. If the converted output is different, but looks correct, runtime regression test as well.

#14 - 03/16/2015 02:42 PM - Ovidiu Maxiniuc

I guess the om_upd20150313b.zip update passed the regression testing. There were a few issues though:

- in CTRL+C the clients were killed twice in ctrlc_11_session3 so there were semaphore issues in ctrlc_11_session1 and ctrlc_11_session4.
- in main tc_job_clock_002 failed beside tc_job_002. This is a known-to fail test.

I will restart the runtime test and hope the tests not executed will have a second chance now.

#15 - 03/17/2015 11:13 AM - Ovidiu Maxiniuc

I guess after the second run, update passed the test.

#16 - 03/17/2015 02:27 PM - Eric Faulhaber

Please commit and distribute the update.

#17 - 03/18/2015 06:03 AM - Ovidiu Maxiniuc

Update committed to bzr as revno 10815 and distributed by mail.

#18 - 03/21/2015 11:21 PM - Eric Faulhaber

- Status changed from WIP to Closed

- % Done changed from 0 to 100

#19 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

Files

integer-function-1.p	975 Bytes	03/10/2015	Ovidiu Maxiniuc
om_upd20150311a.zip	13.3 KB	03/11/2015	Ovidiu Maxiniuc
om_upd20150313b.zip	12.3 KB	03/13/2015	Ovidiu Maxiniuc