Base Language - Bug #4484

Accumulator confusion using same field for count and total

12/17/2019 02:55 PM - Roger Borrello

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

Description

History

#2 - 12/17/2019 03:02 PM - Roger Borrello

Testcase

20191231 Update: Fixed in 4207a-11366

Checked in: uast/accum_overlap_count-total.p

```
define temp-table tt
   field f1 as character format "x(8)"
   field f2 as decimal format "->>>>>9.99".
create tt. tt.f1 = "1". tt.f2 = 0.
create tt. tt.f1 = "2". tt.f2 = 0.

for each tt
   break by f1:
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (total by f1).
end.
message "Done.".
```

Output from java compiler

```
    [javac] testcases/uast/uast/buildarea/src/com/goldencode/testcases/abl/AccumOverlapCountTotal.java:27: err
or: cannot find symbol
    [javac]       final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
    [javac]                                                                  ^
    [javac]   symbol:   variable fieldRef0
    [javac]   location: class AccumOverlapCountTotal
```

java emitted

```
   public void execute()
   {
      final CountAccumulator accumCount0 = new CountAccumulator();
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);

      externalProcedure(AccumOverlapCountTotal.this, new Block((Body) () ->
```

Just to show what happens when you introduce a new field f3 to receive the total accumulator:

```
   public void execute()
   {
      final CountAccumulator accumCount0 = new CountAccumulator();
      FieldReference fieldRef1 = new FieldReference(tt, "f3");
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef1);

      externalProcedure(AccumOverlapCountTotal.this, new Block((Body) () ->
```

#3 - 12/17/2019 03:32 PM - Constantin Asofiei

This might be related to COUNT which doesn't necessarily need the field reference; so, please expand your test to replace the COUNT accumulator with something else, and see if fieldRef is still emitted.

My assumption is that there is some rule somewhere in TRPL, which says 'do not emit the expression for a COUNT accumulator', but it doesn't track if this expression (i.e. field) is used in some other accumulator.

#4 - 12/17/2019 03:39 PM - Roger Borrello

Constantin Asofiei wrote:

> This might be related to COUNT which doesn't necessarily need the field reference; so, please expand your test to replace the COUNT accumulator with something else, and see if fieldRef is still emitted.
>
> My assumption is that there is some rule somewhere in TRPL, which says 'do not emit the expression for a COUNT accumulator', but it doesn't track if this expression (i.e. field) is used in some other accumulator.

I updated the testcase to use AVERAGE and it didn't mind at all!

```
define temp-table tt
   field f1 as character format "x(8)"
   field f2 as decimal format "->>>>>9.99".
create tt. tt.f1 = "1". tt.f2 = 0.
create tt. tt.f1 = "2". tt.f2 = 0.

for each tt
   break by f1:
      accumulate tt.f2 (average by f1).
      accumulate tt.f2 (total by f1).
end.
message "Done.".
```

This emitted:

```
   public void execute()
   {
      FieldReference fieldRef0 = new FieldReference(tt, "f2");
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);

      externalProcedure(AccumOverlapCountTotal.this, new Block((Body) () ->
      {
         RecordBuffer.openScope(tt);
         tt.create();
         tt.setF1(new character("1"));
         tt.setF2(new integer(0));
         tt.create();
```

```
         tt.setF1(new character("2"));
         tt.setF2(new integer(0));

         PresortQuery query0 = new PresortQuery();
         FieldReference byExpr0 = new FieldReference(tt, "f1");

         forEach("loopLabel0", new Block((Init) () ->
         {
            query0.initialize(tt, ((String) null), null, "tt.id asc");
            query0.enableBreakGroups();
            query0.setNonScrolling();
            query0.addSortCriterion(byExpr0);
            query0.addAccumulator(accumAvg0);
            accumAvg0.addBreakGroup(byExpr0);
            accumAvg0.reset();
            query0.addAccumulator(accumTotal0);
            accumTotal0.addBreakGroup(byExpr0);
            accumTotal0.reset();
         },
         (Body) () ->
         {
            query0.next();
            accumAvg0.accumulate();
            accumTotal0.accumulate();
         }));

         message("Done.");
      }));
   }
}
```

#5 - 12/17/2019 03:46 PM - Constantin Asofiei

OK, I think this confirms the problem is with COUNT. Look at process_aggregate function in annotations/accumulate.rules, there might be a problem there; and the referenced annotation in the AST.

#6 - 12/17/2019 03:59 PM - Roger Borrello

Constantin Asofiei wrote:

> OK, I think this confirms the problem is with COUNT. Look at process_aggregate function in annotations/accumulate.rules, there might be a problem there; and the referenced annotation in the AST.

Just to further that theory, as I start to look at the rules... as long as COUNT is not first, we are good:

```
      accumulate tt.f2 (total by f1).
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (average by f1).

      FieldReference fieldRef0 = new FieldReference(tt, "f2");
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
      final CountAccumulator accumCount0 = new CountAccumulator();
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
```

```
      accumulate tt.f2 (average by f1).
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (total by f1).

      FieldReference fieldRef0 = new FieldReference(tt, "f2");
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
      final CountAccumulator accumCount0 = new CountAccumulator();
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
```

Both emit compilable code. Put COUNT first...

```
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (average by f1).
      accumulate tt.f2 (total by f1).

      final CountAccumulator accumCount0 = new CountAccumulator();
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
```

The process_aggregate function certainaly seems to be the place to focus, but I'm not sure if the fix would just be to annotate for COUNT as well.

```
         <!-- annotate the expression to indicate it is being referenced by
              an ACCUMULATE statement;  we DO NOT make this annotation if
              the aggregation type is COUNT, because this type does not
              actually use the expression, so it will not be necessary to
              emit the expression if it is only referenced by COUNT types -->
         <expression>
            expr.getAncestor(-1, prog.expression).putAnnotation("referenced", referenced)
         </expression>
```

#7 - 12/17/2019 04:09 PM - Roger Borrello

This looks like the culprit...

```
        <!-- an ACCUMULATE statement expression initially is considered referenced if either it
             already has been marked as referenced by a sibling AGGREGATE child node, or if it
             is not used within a count or sub-count accumulation; this may be overridden below
             if we determine an accumulator object already is defined in the same scope with
             the same expression -->
        <expression>
           referenced = (exprRef.isAnnotation("referenced") and
                         (#(boolean) exprRef.getAnnotation("referenced"))) or
                        (type != prog.kw_count and type != prog.kw_sub_cnt)
        </expression>
```

Would it be simple as removing the check for prog.kw_count? It would seem we'd just create a reference when it may not be needed.

#8 - 12/17/2019 04:26 PM - Roger Borrello

Roger Borrello wrote:

> This looks like the culprit...
>
> [...]
>
> Would it be simple as removing the check for prog.kw_count? It would seem we'd just create a reference when it may not be needed.

It wasn't that simple, because it no longer created a reference to the field when I removed the or (type != prog.kw_count and type != prog.kw_sub_cnt) as all the other cases no longer fulfilled the criteria. Just setting referenced = true works, but I definitely need more information on the requirements for process_aggregate.

#9 - 12/17/2019 04:58 PM - Roger Borrello

Roger Borrello wrote:

> Roger Borrello wrote:
>
> > This looks like the culprit...

> [...]
>
> Would it be simple as removing the check for prog.kw_count? It would seem we'd just create a reference when it may not be needed.

It wasn't that simple, because it no longer created a reference to the field when I removed the or (type != prog.kw_count and type != prog.kw_sub_cnt) as all the other cases no longer fulfilled the criteria. Just setting referenced = true works, but I definitely need more information on the requirements for process_aggregate.

The only place where the private function process_aggregate is called is when we are processing:

- type prog.kw_count or type prog.kw_sub_cnt
- type prog.kw_average or type prog.kw_sub_avg
- type prog.kw_max or type prog.kw_sub_max
- type prog.kw_min or type prog.kw_sub_min
- type prog.kw_total or type prog.kw_sub_tot

I believe it is safe to add the reference in all those circumstances. The override to false is done if the name has already been referenced.

#10 - 12/18/2019 11:04 AM - Roger Borrello

Doing some research related to what makes these aggregate functions *order-dependent*.

Run 1

```
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (average by f1).
      accumulate tt.f2 (total by f1).
```

Run 2

```
      accumulate tt.f2 (average by f1).
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (total by f1).
```

There was no difference between the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_TOTAL section for run 1 and 2. Understandable, since nothing changed in relation to the runs.

Comparing run 1 to run 2 of the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_COUNT, where count was in the 1st position on run 1, and 2nd position on run 2:

- STATEMENT/KW_ACCUM/EXPRESSION has a **peerid** annotation, which doesn't appear in that section of when it was in the 2nd position.

Comparing run 1 to run 2 of the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_AVERAGE, where average was in the 2nd

position on run 1, and 1st position on run 2:

- STATEMENT/KW_ACCUM/EXPRESSION/FIELD_DEC has a **peerid** annotation on the 2nd run, when it was in the 1st position.

Interesting that there is a **peerid** annotation for each when it was in the 1st position, but for count, it was on the EXPRESSION node, but for average it was on the field.

Regarding the EXPRESSION annotations, they were as expected. Regardless of the position of the accumulator, count had *referenced=false* for *javaname=fieldRef0* whether it was first or second, while average had *referenced=true* for *javaname=fieldRef0* whether it was first or second.

So the main difference seems to be where the *peerid* annotation is attached.

#11 - 12/18/2019 01:01 PM - Roger Borrello

Roger Borrello wrote:

> Doing some research related to what makes these aggregate functions *order-dependent*.
>
> Run 1
> [...]
>
> Run 2
> [...]
>
> There was no difference between the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_TOTAL section for run 1 and 2. Understandable, since nothing changed in relation to the runs.
>
> Comparing run 1 to run 2 of the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_COUNT, where count was in the 1st position on run 1, and 2nd position on run 2:
>
> - STATEMENT/KW_ACCUM/EXPRESSION has a **peerid** annotation, which doesn't appear in that section of when it was in the 2nd position.
>
> Comparing run 1 to run 2 of the AST STATEMENT/KW_ACCUM/EXPRESSION AGGREGATE/KW_AVERAGE, where average was in the 2nd position on run 1, and 1st position on run 2:
>
> - STATEMENT/KW_ACCUM/EXPRESSION/FIELD_DEC has a **peerid** annotation on the 2nd run, when it was in the 1st position.
>
> Interesting that there is a **peerid** annotation for each when it was in the 1st position, but for count, it was on the EXPRESSION node, but for average it was on the field.
>
> Regarding the EXPRESSION annotations, they were as expected. Regardless of the position of the accumulator, count had *referenced=false* for *javaname=fieldRef0* whether it was first or second, while average had *referenced=true* for *javaname=fieldRef0* whether it was first or second.
>
> So the main difference seems to be where the *peerid* annotation is attached.

OK... so that was a red herring (thanks, Greg, for keeping out of that rabbit hole). However, in looking at the actions of convert/database_references.rules it looks like this line is preventing a constructor for the **FieldReference** from being emitted:

```
                  <!-- a field reference child of an ACCUMULATE statement expression that is not
                       referenced (for example, a count or sub-count aggregate or an expression
                       that is identical to an accumulator already constructed in the same
                       scope) should not be emitted -->
                  <rule>this.getAncestor(-1, prog.expression, "referenced", false) != null
                          <action>printfln("database_reference.walk_rules: We are being suppressed! javaname=%
s", javaname)</action>
                     <action>suppress = true</action>
                  </rule>
```

Where can I learn more about getAncestor? It may need to be smarter about whether there are any other nodes that need to reference it.

#12 - 12/18/2019 01:02 PM - Constantin Asofiei

Roger, is this limited only for fields? Does FWD convert OK if you replace the field with some variable?

#13 - 12/18/2019 01:06 PM - Roger Borrello

Constantin Asofiei wrote:

> Roger, is this limited only for fields? Does FWD convert OK if you replace the field with some variable?

Is that even possible? How could you aggregate a variable? Would it be an extent?

#14 - 12/18/2019 01:09 PM - Constantin Asofiei

This works fine in 4GL:

```
def var i as int.
repeat i = 1 to 10:
      accumulate i (average).
      accumulate i (total).
end.
```

#15 - 12/18/2019 01:20 PM - Roger Borrello

Constantin Asofiei wrote:

> This works fine in 4GL:
> [...]

I added count...

```
def var i as int.
repeat i = 1 to 10:
      accumulate i (count).
      accumulate i (average).
      accumulate i (total).
end.
message "Done.".
```

It converts/compiles fine. The variable is defined, versus the field reference using a database.

```
public class Doit
{
   integer i = UndoableFactory.integer();
```

```
   /**
    * External procedure (converted to Java from the 4GL source code
    * in abl/doit.p).
    */
   public void execute()
   {
      final CountAccumulator accumCount0 = new CountAccumulator();
      final AverageAccumulator accumAvg0 = new AverageAccumulator(i);
      final TotalAccumulator accumTotal0 = new TotalAccumulator(i);

      externalProcedure(Doit.this, new Block((Body) () ->
      {
         ToClause toClause0 = new ToClause(i, 1, 10);

         repeatTo("loopLabel0", toClause0, new Block((Init) () ->
         {
            accumCount0.reset();
            accumAvg0.reset();
            accumTotal0.reset();
         },
         (Body) () ->
         {
            accumCount0.accumulate();
            accumAvg0.accumulate();
            accumTotal0.accumulate();
         }));

         message("Done.");
      }));
   }
}
```

The scope of the variable is very difference from field reference in the database version:

```
   public void execute()
   {
      FieldReference fieldRef0 = new FieldReference(tt, "f2");
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
      final CountAccumulator accumCount0 = new CountAccumulator();
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
```

#16 - 12/18/2019 01:24 PM - Roger Borrello

Roger Borrello wrote:

> Constantin Asofiei wrote:
>
> > This works fine in 4GL:
> > [...]
>
> I added count...
> [...]
>
> It converts/compiles fine. The variable is defined, versus the field reference using a database.
>
> [...]
>
> The scope of the variable is very difference from field reference in the database version:
>
> [...]

Interesting... if we have nothing but count, it still declares i:

```
public class Doit
{
   integer i = UndoableFactory.integer();

   /**
    * External procedure (converted to Java from the 4GL source code
    * in abl/doit.p).
    */
   public void execute()
   {
      final CountAccumulator accumCount0 = new CountAccumulator();

      externalProcedure(Doit.this, new Block((Body) () ->
      {
         ToClause toClause0 = new ToClause(i, 1, 10);

         // accumulate i (average).
         repeatTo("loopLabel0", toClause0, new Block((Init) () ->
         {
            accumCount0.reset();
         },
         (Body) () ->
         {
            accumCount0.accumulate();
         }));

         // accumulate i (total).

         message("Done.");
      }));
   }
}
```

#17 - 12/18/2019 01:25 PM - Constantin Asofiei

OK, try this, too:

```
def var i as int.
repeat i = 1 to 10:
      accumulate i + 1 (count).
      accumulate i + 1 (average).
      accumulate i + 1 (total).
end.
```

Is normal to declare i, as is an explicit var. I'm trying to understand if this is an issue limited to fields or any kind of accumulator expression.

#18 - 12/18/2019 01:29 PM - Roger Borrello

Constantin Asofiei wrote:

> OK, try this, too:
> [...]
>
> Is normal to declare i, as is an explicit var. I'm trying to understand if this is an issue limited to fields or any kind of accumulator expression.

I tried that immediately above. It is an issue limited to fields. Are we trying to be too *fine* and only construct a field reference for non count/subcount? I will look into this. Is it generally known the expense of instantiating a field reference it too big a price to take for all the customers that are using count accumulators on fields? We seem to be taking it for variables.

#19 - 12/18/2019 01:34 PM - Constantin Asofiei

Roger Borrello wrote:

> I tried that immediately above. It is an issue limited to fields. Are we trying to be too *fine* and only construct a field reference for non count/subcount? I will look into this. Is it generally known the expense of instantiating a field reference it too big a price to take for all the customers that are using count accumulators on fields? We seem to be taking it for variables.

With variables, is normal and required to exist only one definition. For fields, we need to use a FieldReference because we can't use the getter (as in tt1.getF1()) - the getter gives us the 'current value', while the accumulator needs to be allowed to resolve the value each time it needs it (i.e. FieldReference is a Resolvable).

The idea behind emitting only one FieldReference, and use that for any accumulator which matches it, was an optimization.

#20 - 12/18/2019 01:37 PM - Roger Borrello

Constantin Asofiei wrote:

> Roger Borrello wrote:
>
> > I tried that immediately above. It is an issue limited to fields. Are we trying to be too *fine* and only construct a field reference for non count/subcount? I will look into this. Is it generally known the expense of instantiating a field reference it too big a price to take for all the customers that are using count accumulators on fields? We seem to be taking it for variables.
>
> With variables, is normal and required to exist only one definition. For fields, we need to use a FieldReference because we can't use the getter (as in tt1.getF1()) - the getter gives us the 'current value', while the accumulator needs to be allowed to resolve the value each time it needs it (i.e. FieldReference is a Resolvable).
>
> The idea behind emitting only one FieldReference, and use that for any accumulator which matches it, was an optimization.

Not a good idea to undo any of that. So we should consider this unique to fields. So I'll go back to learning about **getAncestor()**.

#21 - 12/18/2019 01:47 PM - Constantin Asofiei

Roger, there's another different in the AST: look for the hidden attribute at the AST node; this depends on the suppress value in annotations/accumulate.rules. I don't think the problem is with getAncestor, but with the fact that the ASTs are hidden.

#22 - 12/18/2019 01:53 PM - Roger Borrello

Constantin Asofiei wrote:

> Roger, there's another different in the AST: look for the hidden attribute at the AST node; this depends on the suppress value in annotations/accumulate.rules. I don't think the problem is with getAncestor, but with the fact that the ASTs are hidden.

I'll investigate. Thank you.

#23 - 12/18/2019 03:25 PM - Roger Borrello

Roger Borrello wrote:

Constantin Asofiei wrote:

> Roger, there's another different in the AST: look for the hidden attribute at the AST node; this depends on the suppress value in annotations/accumulate.rules. I don't think the problem is with getAncestor, but with the fact that the ASTs are hidden.

I'll investigate. Thank you.

I gave it some investigation...

Node	AVG First	CNT First
STATEMENT/KW_ACCUM/EXPRESSION	**Hidden not set**	**Hidden not set**
STATEMENT/KW_ACCUM/EXPRESSION	**referenced=true**	**referenced=false**

Since there isn't any difference between hidden settings, how could it help? We need to be able to go back to the parent BLOCK, then see if any STATEMENT/KW_ACCUM/EXPRESSION have **referenced=true**. If so, either they should all have it, or at least the first child should.

#24 - 12/18/2019 03:34 PM - Constantin Asofiei

Check the other KW_ACCUM/EXPRESSION nodes. Once COUNT is encountered, any STATEMENT/KW_ACCUM/EXPRESSION will have hidden="true". So:

- if COUNT is first, the expression is not emitted as all subsequent expressions are hidden, too.
- if COUNT is **not** first, then at least one case will not be hidden, and the expression will be emitted.

#25 - 12/18/2019 04:08 PM - Roger Borrello

Constantin Asofiei wrote:

> Check the other KW_ACCUM/EXPRESSION nodes. Once COUNT is encountered, any STATEMENT/KW_ACCUM/EXPRESSION will have hidden="true". So:
>
> - if COUNT is first, the expression is not emitted as all subsequent expressions are hidden, too.
> - if COUNT is **not** first, then at least one case will not be hidden, and the expression will be emitted.

I see what you are saying, but wouldn't we need an annotations/accumulate_post.rules to:

1. Stop at BLOCK/STATEMENT/KW_ACCUM
2. Find first KW_ACCUM/AGGREGATE child

3. If first child **is** prog.kw_count or prog.kw_sub_cnt
    - Find all KW_ACCUM/AGGREGATE children with **exprname** = KW_ACCUM/EXPRESSION (**javaname**)
    - Set KW_ACCUM/EXPRESSION (**referenced** = **true**)

#26 - 12/18/2019 04:24 PM - Constantin Asofiei

Roger, please check this patch:

```
### Eclipse Workspace Patch 1.0
#P p2j
Index: rules/annotations/accumulate.rules
===================================================================
--- rules/annotations/accumulate.rules    (revision 2213)
+++ rules/annotations/accumulate.rules    (working copy)
@@ -428,13 +428,13 @@
                  aggregate types are count or sub-count -->
             <rule on="false">evalLib("fieldtype", ref.type)
                <rule>javaname == null
-                  <action>suppress = true</action>
+                  <action>suppress = false</action>
                   <action>aggref = this.nextSibling</action>
                   <rule>aggref != null
                      <action>childref = aggref.firstChild</action>
                      <while>childref != null
-                        <rule>childref.type != prog.kw_count and type != prog.kw_sub_cnt
-                           <action>suppress = false</action>
+                        <rule>childref.type == prog.kw_count or type == prog.kw_sub_cnt
+                           <action>suppress = true</action>
                            <break/>
                            <action on="false">childref = childref.nextSibling</action>
                         </rule>
```

The problem was that the check if the expression needs to be suppressed or not was incorrect: it needs to look if there are COUNT or SUB-COUNT nodes. Instead, it was looking if there is at least a node which is not COUNT or SUB-COUNT - as it was finding the BY clause, it was emitting a javaname for this expression, which messed up the exprNames dictionary, and which lead to the incorrect assumption 'there is a javaname for this expression, then it means it was already emitted, so hide the AST' (see <action on="false">copy.setHidden(true)</action> on line 448).

#27 - 12/18/2019 04:34 PM - Roger Borrello

Constantin Asofiei wrote:

> Roger, please check this patch:
> [...]
>
> The problem was that the check if the expression needs to be suppressed or not was incorrect: it needs to look if there are COUNT or SUB-COUNT nodes. Instead, it was looking if there is at least a node which is not COUNT or SUB-COUNT - as it was finding the BY clause, it was emitting a javaname for this expression, which messed up the exprNames dictionary, and which lead to the incorrect assumption 'there is a javaname for this expression, then it means it was already emitted, so hide the AST' (see <action on="false">copy.setHidden(true)</action> on line 448).

Yes! That will be my early Christmas present!

```
      final CountAccumulator accumCount0 = new CountAccumulator();
      FieldReference fieldRef0 = new FieldReference(tt, "f2");
      final AverageAccumulator accumAvg0 = new AverageAccumulator(fieldRef0);
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
```

#28 - 12/18/2019 04:51 PM - Roger Borrello

Update to rules/annotations/accumulate.rules checked into 4207a-11358.

#29 - 12/18/2019 04:52 PM - Roger Borrello

- Status changed from New to WIP

#30 - 12/19/2019 07:12 PM - Roger Borrello

Found a subsequent error, when multiple accumulators are on the same statement.

Added a new testcase uast/accum_single_line_count-total.p

There are many permutations possible, but as long as COUNT is in the mix, there will be a missing FieldReference.

#31 - 12/20/2019 09:31 AM - Constantin Asofiei

Roger, see 4207a rev 11362, it should fix the issue.

#32 - 12/20/2019 10:00 AM - Roger Borrello

Constantin Asofiei wrote:

> Roger, see 4207a rev 11362, it should fix the issue.

I'll take a look

#33 - 12/20/2019 10:02 AM - Roger Borrello

Roger Borrello wrote:

> Constantin Asofiei wrote:
>
> > Roger, see 4207a rev 11362, it should fix the issue.
>
> I'll take a look

There's a typo in there, as <action>supress = false</action should be <action>suppress = false</action

I will fix that. :-)

#34 - 12/20/2019 12:12 PM - Roger Borrello

Just reporting that there is another scenario which creates a hidden FieldRef. I checked in uast/accum/accum_count_hiding_fieldref.p

```
define temp-table tt
   field f1 as character format "x(8)"
   field f2 as decimal format "->>>>>9.99"
   field f3 as integer.
create tt. tt.f1 = "1". tt.f2 = 0. tt.f3 = 1.
create tt. tt.f1 = "2". tt.f2 = 0. tt.f3 = 1.

/* As long as COUNT is first, it will break */
for each tt
   break by f1:
      accumulate tt.f2 (count by f1).
      accumulate tt.f2 (total by f1).
      accumulate tt.f3 (total by f1).
end.

message "Done.".
```

This results in

```
      final CountAccumulator accumCount0 = new CountAccumulator();
      final TotalAccumulator accumTotal0 = new TotalAccumulator(fieldRef0);
      FieldReference fieldRef1 = new FieldReference(tt, "f3");
```

```
      final TotalAccumulator accumTotal1 = new TotalAccumulator(fieldRef1);
```

#35 - 12/20/2019 12:45 PM - Constantin Asofiei

Argh, there was another typo. fixed in 11366

#36 - 12/20/2019 01:09 PM - Roger Borrello

Constantin Asofiei wrote:

> Argh, there was another typo. fixed in 11366

Great... this update worked! We need to take accumulate out back for a good beating! :-)

The affected customer files now compile, too!

#37 - 12/31/2019 12:00 PM - Greg Shah

- Status changed from WIP to Test

- % Done changed from 0 to 100

#38 - 01/14/2020 02:12 PM - Roger Borrello

- Status changed from Test to WIP

#39 - 01/14/2020 02:17 PM - Roger Borrello

The code no longer creates a field reference when there is a [downpath:

```
define temp-table tt field f1 as int extent 2.
create tt. tt.f1[1] = 0. tt.f1[2] = 1.
create tt. tt.f1[1] = 0. tt.f1[2] = 1.

for each tt
   break by f1:
      accumulate tt.f1[1] (count).
      accumulate tt.f1[1] (total).
      //accumulate tt.f1[2] (total).
end.

message "Done.".
```

You get

```
    [javac] testcases/src/com/goldencode/testcases/accum/AccumSimple.java:27: error: cannot find symbol
    [javac]       final TotalAccumulator accumTotal0 = new TotalAccumulator(accumExpr0);
    [javac]                                                                 ^
    [javac]   symbol:   variable accumExpr0
    [javac]   location: class AccumSimple
    [javac] 1 error
```

#40 - 01/14/2020 03:57 PM - Roger Borrello

Roger Borrello wrote:

> The code no longer creates a field reference when there is a [downpath:
>
> [...]
>
> You get
> [...]

Constantin, what was the testcase you were working with for validating this change to rules/annotations/accumulate.rules:

```
**     CA  20191219          A subscripted field/var in an accumulator must convert as an
**                           expression.
```

I remove the and !ref.downPath("LBRACKET") condition, and my new testcase works.

Checked in uast/accum/accum_count_and_total_for_extent_field.p

#41 - 01/15/2020 01:50 PM - Roger Borrello

Roger Borrello wrote:

> Roger Borrello wrote:
>
> > The code no longer creates a field reference when there is a [downpath:
> >
> > [...]
> >
> > You get
> > [...]
>
> Constantin, what was the testcase you were working with for validating this change to rules/annotations/accumulate.rules:
> [...]
>
> I remove the and !ref.downPath("LBRACKET") condition, and my new testcase works.
>
> Checked in uast/accum/accum_count_and_total_for_extent_field.p

Hi Constantin... fyi my regression testing was successful with that code removed. What was the reason for adding it?

#42 - 01/16/2020 01:42 PM - Constantin Asofiei

Roger Borrello wrote:

> Hi Constantin... fyi my regression testing was successful with that code removed. What was the reason for adding it?

The reason was a miss-understanding of the initial issue... remember the subsequent revisions which further improved your scenarios? This change should not have survived them. So please commit this change and remove my history entry, as is incorrect.

#43 - 01/16/2020 02:50 PM - Roger Borrello

Constantin Asofiei wrote:

> Roger Borrello wrote:
>
> > Hi Constantin... fyi my regression testing was successful with that code removed. What was the reason for adding it?
>
> The reason was a miss-understanding of the initial issue... remember the subsequent revisions which further improved your scenarios? This change should not have survived them. So please commit this change and remove my history entry, as is incorrect.

Thank you. Committed to **4207a-11380**

#44 - 03/03/2020 02:54 PM - Roger Borrello

- Assignee set to Roger Borrello

- Status changed from WIP to Review

Task branch 4207a was merged to trunk as revision 11344. Task can be moved to closed. It was still in WIP.

#45 - 03/04/2020 10:31 AM - Greg Shah

- Status changed from Review to Closed