Database - Bug #4426

Cannot find parent when processing REPEAT PRESELECT

11/22/2019 04:05 PM - Roger Borrello

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

History

#1 - 11/22/2019 04:17 PM - Roger Borrello

General

Expression execution error when there is a preselect qualifier on repeat.

20191231 Update: Customer code was modified to work around. Fix still pending.
Update: ECF is looking at this

P2J Branch

This is being handled in Branch **4207a**

Testcase

Checked into testcase_repo:
uast/cannot_graft_repeat_preselect.p

```
define temp-table tt
  field num as integer.
create tt.
tt.num = 0.
def var id as integer.
repeat preselect
    each tt where tt.num = id:

    find next tt where tt.num = id.
end.
message "Done.".
```

Artifacts

Execution Log

```
     [java] Core Code Conversion
     [java] --------------------------------------------------------------------------------
     [java]
     [java] Optional rule set [customer_specific_conversion] not found.
     [java] ./abl/cannot_graft_repeat_preselect.p
     [java] EXPRESSION EXECUTION ERROR:
     [java] --------------------------
     [java] tw.graft("anon_array", copy, location, "classname", "Object", "extent", string(numImmediateChildre
n))
     [java]    ^  { Invalid parentid:  -1 }
     [java] --------------------------
```

```
     [java] ERROR:
     [java] com.goldencode.p2j.pattern.TreeWalkException: ERROR!  Active Rule:
     [java] -----------------------
     [java]       RULE REPORT
     [java] -----------------------
     [java] Rule Type :   WALK
     [java] Source AST:  [  ] BLOCK/INNER_BLOCK/BLOCK/STATEMENT/KW_FIND/RECORD_PHRASE/QUERY_SUBST/ @0:0 {16320
8757349}
     [java] Copy AST  :  [  ] BLOCK/INNER_BLOCK/BLOCK/STATEMENT/KW_FIND/RECORD_PHRASE/QUERY_SUBST/ @0:0 {16320
8757349}
     [java] Condition :  tw.graft("anon_array", copy, location, "classname", "Object", "extent", string(numImm
ediateChildren))
     [java] Loop      :  false
     [java] --- END RULE REPORT ---
     [java]
     [java]
     [java]
     [java]     at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1070)
     [java]     at com.goldencode.p2j.convert.TransformDriver.processTrees(TransformDriver.java:542)
     [java]     at com.goldencode.p2j.convert.ConversionDriver.back(ConversionDriver.java:580)
     [java]     at com.goldencode.p2j.convert.TransformDriver.executeJob(TransformDriver.java:876)
     [java]     at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:983)
     [java] Caused by: com.goldencode.expr.ExpressionException: Expression execution error @1:4 [QUERY_SUBST i
d=163208757349]
     [java]     at com.goldencode.p2j.pattern.AstWalker.walk(AstWalker.java:275)
     [java]     at com.goldencode.p2j.pattern.AstWalker.walk(AstWalker.java:210)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.apply(PatternEngine.java:1633)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1531)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1479)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1034)
     [java]     ... 4 more
     [java] Caused by: com.goldencode.expr.ExpressionException: Expression execution error @1:4
     [java]     at com.goldencode.expr.Expression.execute(Expression.java:484)
     [java]     at com.goldencode.p2j.pattern.Rule.apply(Rule.java:497)
     [java]     at com.goldencode.p2j.pattern.Rule.executeActions(Rule.java:745)
     [java]     at com.goldencode.p2j.pattern.Rule.coreProcessing(Rule.java:712)
     [java]     at com.goldencode.p2j.pattern.Rule.apply(Rule.java:534)
     [java]     at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:585)
     [java]     at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:98)
     [java]     at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:585)
     [java]     at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:98)
     [java]     at com.goldencode.p2j.pattern.AstWalker.walk(AstWalker.java:262)
     [java]     ... 9 more
     [java] Caused by: java.lang.IllegalArgumentException: Invalid parentid:  -1
     [java]     at com.goldencode.p2j.pattern.TemplateWorker$Template.graftAt(TemplateWorker.java:365)
     [java]     at com.goldencode.p2j.pattern.TemplateWorker$Template.graft(TemplateWorker.java:308)
     [java]     at com.goldencode.expr.CE12351.execute(Unknown Source)
     [java]     at com.goldencode.expr.Expression.execute(Expression.java:391)
     [java]     ... 18 more
     [java] Elapsed job time:  00:00:00.841

BUILD FAILED
```

AST Snippet

```
     <ast col="1" id="163208757295" line="6" text="repeat" type="KW_REPEAT">
       <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
       <annotation datatype="java.lang.Boolean" key="infinite_loop" value="true"/>
       <annotation datatype="java.lang.Long" key="table_count" value="1"/>
       <annotation datatype="java.lang.Boolean" key="database_access" value="true"/>
       <annotation datatype="java.lang.Boolean" key="explicit_sort" value="false"/>
       <annotation datatype="java.lang.String" key="uber_order_by" value="tt.id asc"/>
       <annotation datatype="java.lang.Long" key="presel_tables" value="1"/>
       <annotation datatype="java.lang.Boolean" key="cross_database" value="false"/>
       <annotation datatype="java.lang.Boolean" key="client_where" value="false"/>
       <annotation datatype="java.lang.String" key="presel_qryname" value="query0"/>
       <ast col="8" id="163208757297" line="6" text="preselect" type="KW_PRESEL">
         <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
         <ast col="5" id="163208757299" line="7" text="each" type="KW_EACH"/>
         <ast col="0" id="163208757301" line="0" text="record phrase" type="RECORD_PHRASE">
           <annotation datatype="java.lang.Boolean" key="index_driven" value="true"/>
           <annotation datatype="java.lang.Boolean" key="sort_overlap" value="false"/>
```

```
          <annotation datatype="java.lang.String" key="order_by" value="tt.id asc"/>
          <annotation datatype="java.lang.Boolean" key="embedded_reference" value="false"/>
          <annotation datatype="java.lang.Boolean" key="preselect" value="true"/>
          <annotation datatype="java.lang.String" key="hql_where" value="tt.num = ?"/>
          <annotation datatype="java.lang.String" key="hql_where_alt" value="tt.num = ?"/>
          <annotation datatype="java.lang.Boolean" key="nested_ref" value="false"/>
          <annotation datatype="java.lang.Boolean" key="contiguous" value="true"/>
          <ast col="10" id="163208757302" line="7" text="tt" type="TEMP_TABLE">
            <annotation datatype="java.lang.String" key="schemaname" value="tt"/>
            <annotation datatype="java.lang.String" key="bufname" value="tt"/>
            <annotation datatype="java.lang.String" key="dbname" value=""/>
            <annotation datatype="java.lang.Long" key="recordtype" value="14"/>
            <annotation datatype="java.lang.String" key="bufrefkey" value="tt,tt,-1"/>
            <annotation datatype="java.lang.Long" key="bufreftype" value="19"/>
            <annotation datatype="java.lang.String" key="uniquename" value="tt_tt"/>
            <annotation datatype="java.lang.Long" key="refid" value="163208757345"/>
          </ast>
          <ast col="13" id="163208757304" line="7" text="where" type="KW_WHERE">
            <annotation datatype="java.lang.Boolean" key="nested_ref" value="false"/>
            <ast col="0" hidden="true" id="163208757306" line="0" text="expression" type="EXPRESSION">
              <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
              <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
              <ast col="26" id="163208757307" line="7" text="=" type="EQUALS">
                <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
                <annotation datatype="java.lang.Long" key="matchtype" value="92"/>
                <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
                <ast col="19" id="163208757310" line="7" text="tt.num" type="FIELD_INT">
                  <annotation datatype="java.lang.Long" key="oldtype" value="6"/>
                  <annotation datatype="java.lang.String" key="schemaname" value="tt.num"/>
                  <annotation datatype="java.lang.String" key="bufname" value="tt"/>
                  <annotation datatype="java.lang.String" key="dbname" value=""/>
                  <annotation datatype="java.lang.Long" key="recordtype" value="14"/>
                  <annotation datatype="java.lang.String" key="name" value="tt.num"/>
                  <annotation datatype="java.lang.Long" key="type" value="410"/>
                  <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
                  <annotation datatype="java.lang.Boolean" key="is_meta" value="false"/>
                  <annotation datatype="java.lang.String" key="fieldname" value="tt.num"/>
                  <annotation datatype="java.lang.String" key="bufrefkey" value="tt,tt,-1"/>
                  <annotation datatype="java.lang.Long" key="bufreftype" value="21"/>
                  <annotation datatype="java.lang.String" key="uniquename" value="tt_tt"/>
                  <annotation datatype="java.lang.Long" key="refid" value="163208757345"/>
                  <annotation datatype="java.lang.String" key="methodtxt" value="getNum"/>
                  <annotation datatype="java.lang.Boolean" key="current_buffer" value="true"/>
                  <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
                </ast>
                <ast col="28" id="163208757311" line="7" text="id" type="VAR_INT">
                  <annotation datatype="java.lang.Long" key="oldtype" value="2781"/>
                  <annotation datatype="java.lang.Long" key="refid" value="163208757283"/>
                  <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
                  <annotation datatype="java.lang.Boolean" key="sub_expression" value="true"/>
                  <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
                </ast>
              </ast>
            </ast>
          </ast>
          <ast col="0" id="163208757346" line="0" text="" type="QUERY_SUBST">
            <ast col="0" id="163208757347" line="0" text="" type="EXPRESSION">
              <annotation datatype="java.lang.Boolean" key="deferred" value="false"/>
              <ast col="28" id="163208757348" line="7" text="id" type="VAR_INT">
                <annotation datatype="java.lang.Long" key="oldtype" value="2781"/>
                <annotation datatype="java.lang.Long" key="refid" value="163208757283"/>
                <annotation datatype="java.lang.Long" key="support_level" value="16400"/>
                <annotation datatype="java.lang.Boolean" key="sub_expression" value="true"/>
                <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
                <annotation datatype="java.lang.Long" key="oriref" value="163208757311"/>
              </ast>
            </ast>
          </ast>
        </ast>
      </ast>
    </ast>
```

#3 - 11/26/2019 01:44 PM - Eric Faulhaber

database_access.rules is one of the nastier persistence rule sets, because it tries to handle so many related but separate use cases. It seems the error is there. I will look at it...

#4 - 12/03/2019 06:09 AM - Eric Faulhaber

So, the problem is that we don't anticipate the where clause for the inner FIND statement. In every other case we've seen of a FIND within a REPEAT PRESELECT, the FIND acts as a simple fetch for the REPEAT statement. As such, we always expect it to be in the form:

```
FIND tt.
```

If you remove the where clause from the FIND in the above test case, it converts without error, into:

```
         ...

         PreselectQuery query0 = new PreselectQuery();

         repeat("loopLabel0", new Block((Init) () ->
         {
            query0.initialize(tt, "tt.num = ?", null, "tt.id asc", new Object[]
            {
               id
            });
         },
         (Body) () ->
         {
            query0.next();
         }));

         ...
```

Currently, we have no API for PreselectQuery.next which accepts an additional filter expression, as I was not aware this syntax was legal.

We could add an API which accepts a client-side where clause in the form of a lambda, to be executed on each result of the containing PreselectQuery, until a match is found or the result set is exhausted. In this case, that lambda expression would be redundant, because the FIND statement's where clause matches the enclosing REPEAT PRESELECT's where clause. Ideally, we would detect this during conversion and just hide the part of the tree which represents the FIND statement's where clause, so the converted code would emit as above.

However, I have confirmed that it is legal to have a non-matching where clause for the FIND. In this case, we would need to emit the lambda as an additional filter.

Either way, this is not a trivial fix to make. If the 4GL code can be patched to remove the FIND statement's where clause, that would be the most expedient solution. However, it is not a viable long term solution; eventually, this oversight must be addressed.

#5 - 12/03/2019 09:47 AM - Roger Borrello

Eric Faulhaber wrote:

> We could add an API which accepts a client-side where clause in the form of a lambda, to be executed on each result of the containing PreselectQuery, until a match is found or the result set is exhausted. In this case, that lambda expression would be redundant, because the FIND statement's where clause matches the enclosing REPEAT PRESELECT's where clause. Ideally, we would detect this during conversion and just hide the part of the tree which represents the FIND statement's where clause, so the converted code would emit as above.
>
> However, I have confirmed that it is legal to have a non-matching where clause for the FIND. In this case, we would need to emit the lambda as an additional filter.

The PRESELECT phrase documentation (https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dvref%2Fpreselect-phrase.html) mentions the situation where the FIND statement could conflict, and seems to suggest it would override the PRESELECT's where clause. Does my assessment seem accurate?

#6 - 12/03/2019 10:16 AM - Roger Borrello

This is by far the most prevalent use of this syntax in customer code (PRESELECT doesn't filter anything, and lets the internal FIND perform indexing):

```
define temp-table tt
  field num as integer.
create tt.
tt.num = 0.
def var id as integer.
repeat preselect
    each tt: /* where tt.num = id:*/

    find next tt where tt.num = id.
end.
message "Done.".
```

#7 - 12/03/2019 02:53 PM - Eric Faulhaber

Which construct controls sorting? That is, if the REPEAT PRESELECT specifies one index and the inner FIND a different one, in what order do the records come back?

#8 - 12/03/2019 03:10 PM - Roger Borrello

Eric Faulhaber wrote:

> Which construct controls sorting? That is, if the REPEAT PRESELECT specifies one index and the inner FIND a different one, in what order do the records come back?

I moved our cannot_graft_repeat_preselect.p testcase to uast/preselect_where/ and added an additional one uast/preselect_where/preselect_list_odds.p to show that in there is not difference in the fact that we may be looking for odd numbers in a table, regardless if the where (tt.num modulo 2) = 1 is on the PRESELECT line, the FIND NEXT line, or both. This accounts for the majority of customer code using this.

I'm working on another testcase to show if the PRESELECT line is very constrictive, relaxing the FIND NEXT shouldn't matter:

```
REPEAT PRESELECT
  EACH tt where (tt.num modulo 2) = 1 AND tt.num > 5
  FIND NEXT tt WHERE (tt.num modulo 2) = 1 NO-ERROR.
END.
```

This should yield the same results it the WHERE is removed from FIND NEXT:

```
REPEAT PRESELECT
  EACH tt where (tt.num modulo 2) = 1 AND tt.num > 5
  FIND NEXT tt NO-ERROR.
END.
```

These are the only 2 situations currently in the code I am dealing with. Of course you are concerned with a much more wide range of situations. You could create a testcase off of one of mine.

#9 - 12/03/2019 03:52 PM - Roger Borrello

Roger Borrello wrote:

> Eric Faulhaber wrote:
>
> > Which construct controls sorting? That is, if the REPEAT PRESELECT specifies one index and the inner FIND a different one, in what order do the records come back?
>
> I moved our cannot_graft_repeat_preselect.p testcase to uast/preselect_where/ and added an additional one uast/preselect_where/preselect_list_odds.p to show that in there is not difference in the fact that we may be looking for odd numbers in a table, regardless if the where (tt.num modulo 2) = 1 is on the PRESELECT line, the FIND NEXT line, or both. This accounts for the majority of customer code using this.
>
> I'm working on another testcase to show if the PRESELECT line is very constrictive, relaxing the FIND NEXT shouldn't matter:
> [...]
>
> This should yield the same results it the WHERE is removed from FIND NEXT:
> [...]
>
> These are the only 2 situations currently in the code I am dealing with. Of course you are concerned with a much more wide range of situations. You could create a testcase off of one of mine.

The 2 testcases are in uast/preselect_where if you get the latest revision. In each of the 2 cases, there are no differences in the results, regardless of the "manhandling" done against the where clauses, with the exception of conflicting where clauses.

preselect_where/preselect_list_restrictive_to_loose.p has that 3rd case, and the result is no available data.

```
procedure testcase3:
   define input parameter table for tt.
   define input parameter id as integer.

   display "   preselect has odd where clause and tt.num >" id skip.
   display "   find next has tt.num <" id skip.
   repeat preselect each tt where (tt.num modulo 2) = 1 and tt.num > id.
      find next tt where tt.num < id no-error.
      if not available tt then do:
        display "No more data".
        leave.
      end.
      display tt.num label "tt.num (3)".
   end.
end procedure.
```

#10 - 12/03/2019 06:57 PM - Eric Faulhaber

- Project changed from Base Language to Database

What I'm trying to figure out is whether it is safe to move the FIND's where clause to the REPEAT PRESELECT level, either as an override or augment. This would allow us to handle the query and all filtering at the database server in one SQL statement, and the FIND would remain a simple fetch (e.g., query0.next()). But I have to understand the role of each of these parts better, as well as the interaction between them, to know whether this is safe to do.

#11 - 12/04/2019 09:49 AM - Roger Borrello

Eric Faulhaber wrote:

> What I'm trying to figure out is whether it is safe to move the FIND's where clause to the REPEAT PRESELECT level, either as an override or augment. This would allow us to handle the query and all filtering at the database server in one SQL statement, and the FIND would remain a simple fetch (e.g., query0.next()). But I have to understand the role of each of these parts better, as well as the interaction between them, to know whether this is safe to do.

Let me throw this out there... I ran conversion on preselect_list_odds.p, which I hadn't done yesterday, as I was only focused on getting output from Progress. It converted fine. So I started to modify the testcase until it would break by making it look more like cannot_graft_repeat_preselect.p. Turns out that you can get cannot_graft_repeat_preselect.p to convert by simply changing id to 1 in the find next tt where tt.num = line.

What do you make of that?

#12 - 12/04/2019 11:47 AM - Eric Faulhaber

Roger Borrello wrote:

> [...]
> What do you make of that?

It converts incorrectly: looks like the where clause of the FIND is not taken into consideration and it converts as if there were none.

#13 - 12/04/2019 12:07 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Roger Borrello wrote:
>
> > [...]
> > What do you make of that?

> It converts incorrectly: looks like the where clause of the FIND is not taken into consideration and it converts as if there were none.

In that last comment, I was referring to the version of cannot_graft_repeat_preselect.p after making the change to compare tt.num with a constant rather than a variable, as you suggested, but preselect_list_odds.p has the same issue: the where clause of the inner FIND statement is being dropped, in all cases.

We are not getting the explicit failure during conversion because the FINDs' where clauses in these cases do not have any elements which are refactored into query substitution parameters. This is the part which causes conversion to break. But even though conversion completes, it is not successful. By chance, those cases where the FIND statement's where clause matches that of the REPEAT PRESELECT's where clause AND in which the where clause has no elements (such as variables) which would be refactored into query substitution parameters, result in a seemingly correct outcome. However, this is by accident more than by design. We have to deal with those FIND statement where clauses explicitly. I haven't looked into the conversion of this part deeply enough to know why they are simply dropped, but this is not necessarily always the correct behavior.

#14 - 12/04/2019 12:14 PM - Roger Borrello

Eric Faulhaber wrote:

> Eric Faulhaber wrote:
>
> > Roger Borrello wrote:
> >
> > > [...]
> > > What do you make of that?
> >
> > It converts incorrectly: looks like the where clause of the FIND is not taken into consideration and it converts as if there were none.
>
> In that last comment, I was referring to the version of cannot_graft_repeat_preselect.p after making the change to compare tt.num with a constant rather than a variable, as you suggested, but preselect_list_odds.p has the same issue: the where clause of the inner FIND statement is being dropped, in all cases.
>
> We are not getting the explicit failure during conversion because the FINDs' where clauses in these cases do not have any elements which are refactored into query substitution parameters. This is the part which causes conversion to break. But even though conversion completes, it is not successful. By chance, those cases where the FIND statement's where clause matches that of the REPEAT PRESELECT's where clause AND in which the where clause has no elements (such as variables) which would be refactored into query substitution parameters, result in a seemingly correct outcome. However, this is by accident more than by design. We have to deal with those FIND statement where clauses explicitly. I haven't looked into the conversion of this part deeply enough to know why they are simply dropped, but this is not necessarily always the correct behavior.

It sounds like this is a *Base Language* issue, related to the conversion, rather than a *Database* issue.

#15 - 12/04/2019 12:29 PM - Eric Faulhaber

Roger Borrello wrote:

> [...]
> It sounds like this is a *Base Language* issue, related to the conversion, rather than a *Database* issue.

One can look at it that way, but historically, we have categorized the conversion of language statements which are about database functionality into the *Database* sub-project, which is why I moved this issue here. The conversion of where clauses and the refactoring of variables in a where clause into query substitution parameters definitely falls into that category.

#16 - 03/03/2020 02:42 PM - Roger Borrello

Task branch 4207a was merged to trunk as revision 11344. However, this task was worked around in customer code, and hasn't been worked on.